Cyclic Implicit Complexity

Curzi, Gianluca; Das, Anupam

DOI:
10.48550/arXiv.2110.01114

Document Version
Other version

Citation for published version (Harvard):
Curzi, G & Das, A 2021 'Cyclic Implicit Complexity' arXiv, pp. 1-53. https://doi.org/10.48550/arXiv.2110.01114

CYCLIC IMPLICIT COMPLEXITY

GIANLUCA CURZI* AND ANUPAM DAS*

Abstract. *Circular* (or *cyclic*) proofs have received increasing attention in recent years, and have been proposed as an alternative setting for studying (co)inductive reasoning. In particular, now several type systems based on circular reasoning have been proposed. However, little is known about the complexity theoretic aspects of circular proofs, which exhibit sophisticated loop structures atypical of more common 'recursion schemes'. This paper attempts to bridge the gap between circular proofs and *implicit computational complexity*. Namely we introduce a circular proof system based on the Bellantoni and Cook's famous safe-normal function algebra, and we identify suitable proof theoretical constraints to characterise the polynomial-time and elementary computable functions.

1. Introduction

Formal proofs are traditionally seen as finite mathematical objects modelling logical or mathematical reasoning. Non-wellfounded (but finitely branching) proofs represent a generalization of the notion of formal proof to an infinitary setting. In non-wellfounded proof theory, special attention is devoted to *regular* (or *circular*) proofs, i.e. those non-wellfounded proofs having only finitely many distinct subproofs. Regular proofs can be turned into finite structures called *cycle normal forms*, usually given as finite trees with additional 'backpointers'. However, regular proofs admit fallacious reasoning. To recover consistency, a standard solution is based on the introduction of non-local correctness criteria, e.g. *progressiveness*, typically checked by Büchi automata on infinite words.

Regular proofs, and their corresponding cycle normal forms, have been employed to reason about modal μ-calculus and fixed-point logics [NW96, DHL06b], induction and coinduction [BS11], Kleene algebra [DP17, DP18], linear logic [BDS16], arithmetic [Das18], system T [KPP21a, Das21], and continuous cut-elimination [Min78, FS13]. In particular, [KPP21a] and [Das21] investigate the computational aspects of regular proofs. Due to their coinductive nature, non-wellfounded proofs are able to define any number-theoretic (partial) function. Definability can be then restricted by combining regularity and progressiveness: the former rules out uncomputable functions, while the latter recovers termination.

Little is known, however, about the complexity-theoretic aspects of regular proofs. Kuperberg, Pinault and Pous [KPP19] introduced a circular proof system based on Kleene algebras and seen as a computational machinery for recognising languages. The system and its affine version capture, respectively, the regular languages and the languages accepted in logarithmic time by random-access Turing

*University of Birmingham
Date: October 5, 2021.

machines. Their work does not consider the cut rule, which allows for a proof-theoretical description of function composition and is the key ingredient to express recursion in a cyclic-theoretic setting.

The present paper aims at bridging the gap between regular proofs and Implicit Computational Complexity (ICC), a branch of computational complexity studying machine-free languages and calculi able to capture a given complexity class without relying on explicit resource bounds. Our starting point is Bellantoni and Cook's function algebra B characterising the polynomial time computable functions (**FPTIME**) using *safe recursion*. Functions of B have shape $f(x_1, ..., x_n; y_1, ..., y_m)$, where the semicolon separates the *normal* arguments $x_1, ..., x_n$ from the *safe* arguments $y_1, ..., y_m$. The idea behind safe recursion is that only normal arguments can be used as recursive parameters, while recursive calls can only appear in safe position. This prevents recursive calls becoming recursive parameters of other previously defined functions. In the spirit of the Curry-Howard paradigm, which identifies proofs and programs, B can be alternatively designed as a sequent calculus proof-system where safe recursion is introduced by a specific inference rule srec. In this system, a two-sorted function $f(x_1, ..., x_n; y_1, ..., y_m)$ is represented by a derivation of the sequent $\Box N, \overset{n}{...}, \Box N, N, \overset{m}{...}, N \Rightarrow N$ where N is the ground type for natural numbers and $\Box N$ is its modal version.

Starting from B we shall consider the non-wellfounded proofs, or *coderivations*, generated by the rules of the subsystem $\mathsf{B}^- := \mathsf{B} \setminus \mathsf{srec}$. The circular proof system CNB is then obtained by considering the regular and progressing coderivations of B^- which satisfy a further criterion, called *safety*. On the one hand, regularity and progressiveness ensure that coderivations of CNB define total computable functions; on the other hand, the latter criterion ensures that the recursion mechanisms defined in this infinitary setting are safe, i.e. the recursive call of a function is never the recursive parameter of the step function.

Despite CNB having only ground types, it is able to define safe recursion schemes that nest recursive calls, a property that typically arises in higher-order recursion. This is in fact a peculiar feature of regular proofs extensively studied by Das in [Das21], who has shown that the number-theoretic functions definable by type level n proofs of a circular version of system T are exactly those ones definable by type level $n + 1$ proofs of T.

In the setting of ICC, Hofmann [Hof97] and Leivant [Lei99] observed that the capability of nesting recursive calls by higher-order safe recursion mechanisms can be used to characterise the elementary time functions (**FELEMENTARY**). In particular, in [Hof97] Hofmann presents the type system SLR (Safe Linear Recursion) as a higher-order version of B. This system has been shown to capture **FPTIME** once a linearity restriction on the higher-order 'safe' recursion operator is imposed, which prevents duplication of the recursive calls, and hence their nesting.

Following [Hof97], we introduce a linearity requirement for CNB that is able to control the interplay between loops and the cut rule, called *left-leaning criterion*. The resulting circular proof system is called CB. Intuitively, CB can be seen as a circular version of B, while CNB can be seen as the circular version of a new function algebra, called NB, which generalises B by permitting nested versions of the safe recursion scheme. In particular, in order to define the nesting of recursive calls, the function algebra for CNB crucially requires the introduction of 'auxiliary functions', i.e. oracles.

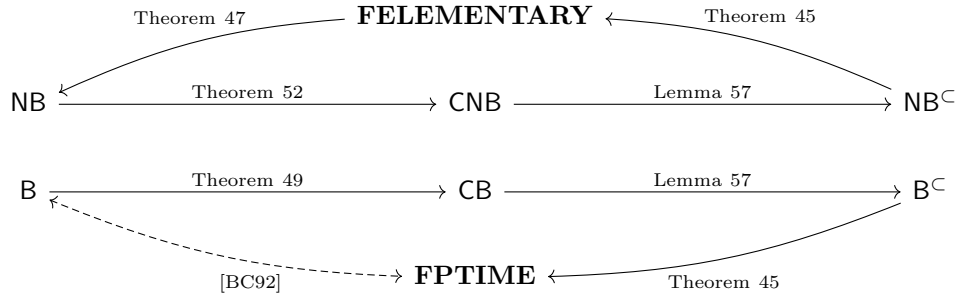


FIGURE 1. Summary of the main results of the paper, where $\to$ indicates an inclusion ($\subseteq$) of function classes.

The main results of our paper are the following:

- a function is definable in CNB iff it is NB iff it is in **FELEMENTARY**;
- a function is definable in CB iff it is in **FPTIME**;

First, we define the function algebras $\mathsf{NB}^{\subset}$ and $\mathsf{B}^{\subset}$, respectively obtained by extending NB and B with a safe recursion mechanism based on permutation of prefixes. Then we show that NB and $\mathsf{NB}^{\subset}$ capture precisely the elementary time computable functions, while $\mathsf{B}^{\subset}$ captures the polytime ones. Completeness for CNB can be achieved by showing how to represent the functions of NB. In a similar way, we represent the functions of B in CB. Soundness is subtler, as it relies on a translation of coderivations of CNB and CB, which are essentially coinductive objects, into the inductively defined functions of $\mathsf{NB}^{\subset}$ and $\mathsf{B}^{\subset}$, respectively.

Overview of results. The paper is structured as follows. Section 2 discusses B as a proof system. In Section 3 we present the non-wellfounded proof system B^{-} and its semantics. We then analyse some proof-theoretical conditions able to restrict the set of definable functions of B^{-}. This leads us to the circular proof systems CNB and CB. In Section 4 we present the function algebras NB, $\mathsf{B}^{\subset}$ and $\mathsf{NB}^{\subset}$, which implement various safe recursion schemes. Section 5 shows that $\mathsf{B}^{\subset}$ captures **FPTIME** (Corollary 46) and that both NB and $\mathsf{NB}^{\subset}$ capture **FELEMENTARY** (Corollary 48). These results require a Bounding Lemma (Lemma 43) and the encoding of the elementary time functions into NB (Theorem 47). In Section 6 we show that any function definable in B is also definable in CB (Theorem 49), and that any function definable in NB is also definable in CNB (Theorem 52). In Section 7 we present a translation of CNB into $\mathsf{NB}^{\subset}$ that maps CB coderivations into $\mathsf{B}^{\subset}$ functions (Lemma 57).

The main results of this section are displayed in Figure 1. Many proofs are reported in the appendices.

2. Preliminaries: two-tiered 'safe-normal' function algebras

Bellantoni and Cook introduced in [BC92] an algebra of functions based on a simple two-sorted structure. This idea was itself inspired by Leivant's impredicative characterisations, one of the founding works in Implicit Computational Complexity (ICC) [Lei91]. The resulting 'tiering' of the underlying sorts has been a recurring theme in the ICC literature since, and so it is this structure that shall form the basis of the systems we consider in this work.

We consider functions on the natural numbers with inputs distinguished into two sorts: 'safe' and 'normal'. We shall write functions explicitly indicating inputs, namely writing $f(x_1, \ldots, x_m; y_1, \ldots, y_n)$ when f takes m normal inputs $\vec{x}$ and n safe inputs $\vec{y}$. Both sorts vary over the natural numbers, but their roles will be distinguished by the closure operations of the algebras and rules of the systems we consider.

Throughout this work, we write $|x|$ for the length of x (in binary notation), and if $\vec{x} = x_1, \ldots, x_m$ we write $|\vec{x}|$ for the list $|x_1|, \ldots, |x_m|$.

2.1. **Bellantoni and Cook's characterisation of polynomial-time functions.** Let us first recall Bellantoni and Cook's algebra in its original guise.

Definition 1 (Bellantoni-Cook algebra)**.** B is the smallest class of (two-sorted) functions containing,

- $0(;) := 0 \in \mathbb{N}$.
- $\pi^{m;n}_{j;}(x_1, \ldots, x_m; y_1, \ldots, y_n) := x_j$, whenever $1 \leq j \leq m$.
- $\pi^{m;n}_{;j}(x_1, \ldots, x_m; y_1, \ldots, y_n) := y_j$, whenever $1 \leq j \leq n$.
- $\mathsf{s}_i(;x) := 2x + i$, for $i \in \{0, 1\}$.
- $\mathsf{p}(;x) := \lfloor x/2 \rfloor$.
- $\mathsf{cond}(;w,x,y,z) := \begin{cases} x & w = 0 \\ y & w = 0 \mod 2, w \neq 0 \\ z & w = 1 \mod 2 \end{cases}$

and closed under the following:

- (Safe composition)
  - If $f(\vec{x}, x; \vec{y}) \in \mathsf{B}$ and $g(\vec{x};) \in \mathsf{B}$ then $f(\vec{x}, g(\vec{x};); \vec{y}) \in \mathsf{B}$.
  - If $f(\vec{x}; \vec{y}, y) \in \mathsf{B}$ and $g(\vec{x}; \vec{y}) \in \mathsf{B}$ then $f(\vec{x}; \vec{y}, g(\vec{x}; \vec{y})) \in \mathsf{B}$.
- (Safe recursion on notation)[1] If $g(\vec{x}; \vec{y}) \in \mathsf{B}$ and $h_i(x, \vec{x}; \vec{y}, y) \in \mathsf{B}$ for $i = 0, 1$ then so is $f(x, \vec{x}; \vec{y})$ given by:

$$\begin{array}{rcll} f(0, \vec{x}; \vec{y}) & := & g(\vec{x}; \vec{y}) & \\ f(\mathsf{s}_0 x, \vec{x}; \vec{y}) & := & h_0(x, \vec{x}; \vec{y}, f(x, \vec{x}; \vec{y})) & \text{if } x \neq 0 \\ f(\mathsf{s}_1 x, \vec{x}; \vec{y}) & := & h_1(x, \vec{x}; \vec{y}, f(x, \vec{x}; \vec{y})) & \end{array}$$

Intuitively, in a function $f(\vec{x}; \vec{y}) \in \mathsf{B}$ only the normal arguments $\vec{x}$ can be used as recursive parameters. The idea behind safe recursion is that recursive calls can only appear in safe position, and hence they can never be used as recursive parameters of other previously defined functions. Safe composition preserves the distinction between normal and safe arguments by requiring that, when composing along a normal parameter, the pre-composing function has no safe parameter at all. As a result, we can move a normal parameter to a safe position but not vice versa.

Writing **FPTIME** for the class of functions computable in polynomial-time, the main result of Bellantoni and Cook is:

Theorem 2 ([BC92])**.** *$f(\vec{x};) \in \mathsf{B}$ if and only if $f(\vec{x}) \in$ **FPTIME**.*

[1]This was originally called 'predicative recursion on notation' by Bellantoni and Cook in [BC92], in reference to Parsons' predicative higher-order characterisation of the primitive recursive functions (à la Kleene).

$$\mathsf{id}\,\frac{}{N \Rightarrow N} \qquad \mathsf{w}_N\,\frac{\Gamma \Rightarrow B}{\Gamma, N \Rightarrow B} \qquad \mathsf{w}_\Box\,\frac{\Gamma \Rightarrow B}{\Box N, \Gamma \Rightarrow B} \qquad \mathsf{e}\,\frac{\Gamma, A, B, \Gamma' \Rightarrow C}{\Gamma, B, A, \Gamma' \Rightarrow C}$$

$$\Box_l\,\frac{\Gamma, N \Rightarrow A}{\Box N, \Gamma \Rightarrow A} \qquad \Box_r\,\frac{\Box\Gamma \Rightarrow N}{\Box\Gamma \Rightarrow \Box N} \qquad 0\,\frac{}{\Rightarrow N} \qquad \mathsf{s}_0\,\frac{\Gamma \Rightarrow A}{\Gamma \Rightarrow A} \qquad \mathsf{s}_1\,\frac{\Gamma \Rightarrow A}{\Gamma \Rightarrow A}$$

$$\mathsf{cut}_N\,\frac{\Gamma \Rightarrow N \quad \Gamma, N \Rightarrow B}{\Gamma \Rightarrow B} \qquad \mathsf{cut}_\Box\,\frac{\Gamma \Rightarrow \Box N \quad \Box N, \Gamma \Rightarrow B}{\Gamma \Rightarrow B}$$

$$\mathsf{cond}_N\,\frac{\Gamma \Rightarrow N \quad \Gamma, N \Rightarrow N \quad \Gamma, N \Rightarrow N}{\Gamma, N \Rightarrow N} \qquad \mathsf{cond}_\Box\,\frac{\Gamma \Rightarrow N \quad \Box N, \Gamma \Rightarrow N \quad \Box N, \Gamma \Rightarrow N}{\Box N, \Gamma \Rightarrow N}$$

$$\mathsf{srec}\,\frac{\Gamma \Rightarrow N \quad \Box N, \Gamma, N \Rightarrow N \quad \Box N, \Gamma, N \Rightarrow N}{\Box N, \Gamma \Rightarrow N}$$

FIGURE 2. System B, as a sequent-style type system.

2.2. **A type theoretic presentation of Bellantoni-Cook.** In this section we recall a well-known formulation of Bellantoni and Cook's algebra as a type system with modalities to distinguish the two sorts (cf., e.g., [Hof97]). In order to facilitate the definition of the circular system that we present later, we here work with sequent-style typing derivations.

We only consider *types* (or *formulas*) N and $\Box N$ which intuitively vary over the natural numbers (we will give a formal semantics soon). We use A, B, etc. to vary over types. A *sequent* is an expression $\Gamma \Rightarrow A$, where Γ is a list of types (called the *context* or *antecedent*) and A is a type (called the *succedent*). For a list of types $\Gamma = N, \overset{k}{\ldots}, N$, we write $\Box\Gamma$ for $\Box N, \overset{k}{\ldots}, \Box N$.

In what follows, we shall essentially identify B with the $S4$-style type system in Figure 2. The colouring of type occurrences may be ignored for now, they will become relevant in the next section. Derivations in this system are simply called B-*derivations*, and will be denoted $\mathcal{D}, \mathcal{E}, \ldots$, and we write $\mathcal{D} : \Gamma \Rightarrow A$ if the derivation $\mathcal{D}$ has end-sequent is $\Gamma \Rightarrow A$.

Convention 3 (Left normal, right safe)**.** In what follows, we shall always assume that sequents have the form $\Box N, \ldots, \Box N, N, \ldots, N \Rightarrow A$, i.e. in the LHS all $\Box N$ occurrences are placed before all N occurrences. Note that this invariant is maintained by the typing rules of Figure 2, as long as we insist that $A = B$ in the exchange rule e. This effectively means that exchange steps have one of the following two forms:

$$\mathsf{e}_N\,\frac{\Gamma, N, N, \vec{N}' \Rightarrow A}{\Gamma, N, N, \vec{N}' \Rightarrow A} \qquad \mathsf{e}_\Box\,\frac{\Box\vec{N}, \Box N, \Box N, \Gamma' \Rightarrow A}{\Box\vec{N}, \Box N, \Box N, \Gamma' \Rightarrow A}$$

Let us point out that this convention does not change the class of definable functions, under the semantics we are about to give, as we are about to define, up to permutation of inputs.

We define this system as a class of safe-normal functions by construing each rule as an operation on safe-normal functions and identifying the system with the class of thus definable functions. Formally:

Definition 4 (Semantics)**.** Given a B-derivation $\mathcal{D} : \Box N, \overset{m}{\ldots}, \Box N, N, \overset{n}{\ldots}, N \Rightarrow A$ we define a two-sorted function $f_{\mathcal{D}}(x_1, \ldots, x_m; y_1, \ldots, y_n)$ by induction on the structure of $\mathcal{D}$ as follows:

- If $\mathcal{D}$ is $\mathsf{id}\dfrac{}{N \Rightarrow N}$ then $f_{\mathcal{D}}(;y) := y$.
- If $\mathcal{D}$ has the form $\mathsf{w}_N\dfrac{\overset{\mathcal{D}_0}{\Gamma \Rightarrow A}}{\Gamma, N \Rightarrow A}$ then $f_{\mathcal{D}}(\vec{x};\vec{y},y) := f_{\mathcal{D}_0}(\vec{x};\vec{y})$.
- If $\mathcal{D}$ is $\mathsf{w}_\Box\dfrac{\overset{\mathcal{D}_0}{\Gamma \Rightarrow A}}{\Box N, \Gamma \Rightarrow A}$ then $f_{\mathcal{D}}(x,\vec{x};\vec{y}) := f_{\mathcal{D}_0}(\vec{x};\vec{y})$.
- If $\mathcal{D}$ is $\mathsf{e}_N\dfrac{\overset{\mathcal{D}_0}{\Gamma, N, N, \vec{N}' \Rightarrow A}}{\Gamma, N, N, \vec{N}' \Rightarrow A}$ then $f_{\mathcal{D}}(\vec{x};\vec{y},y,y',\vec{y}') := f_{\mathcal{D}_0}(\vec{x};\vec{y},y',y,\vec{y}')$.
- If $\mathcal{D}$ is $\mathsf{e}_\Box\dfrac{\overset{\mathcal{D}_0}{\Box\vec{N}, \Box N, \Box N, \Delta \Rightarrow A}}{\Box\vec{N}, \Box N, \Box N, \Delta \Rightarrow A}$ then $f_{\mathcal{D}}(\vec{x},x,x',\vec{x}';\vec{y}) := f_{\mathcal{D}_0}(\vec{x},x',x,\vec{x}';\vec{y})$.
- If $\mathcal{D}$ is $\Box_l\dfrac{\overset{\mathcal{D}_0}{\Gamma, N \Rightarrow A}}{\Box N, \Gamma \Rightarrow A}$ then $f_{\mathcal{D}}(x,\vec{x};\vec{y}) := f_{\mathcal{D}_0}(\vec{x};\vec{y},x)$.
- If $\mathcal{D}$ is $\Box_r\dfrac{\overset{\mathcal{D}_0}{\Box\Gamma \Rightarrow N}}{\Box\Gamma \Rightarrow \Box N}$ then $f_{\mathcal{D}}(\vec{x};) := f_{\mathcal{D}_0}(\vec{x};)$.
- If $\mathcal{D}$ is $0\dfrac{}{\Rightarrow N}$ then $f_{\mathcal{D}}(;) := 0$.
- If $\mathcal{D}$ is $\mathsf{s}_0\dfrac{\overset{\mathcal{D}_0}{\Gamma \Rightarrow A}}{\Gamma \Rightarrow A}$ then $f_{\mathcal{D}}(\vec{x};\vec{y}) := \mathsf{s}_0(;f_{\mathcal{D}_0}(\vec{x};\vec{y}))$.
- If $\mathcal{D}$ is $\mathsf{s}_1\dfrac{\overset{\mathcal{D}_0}{\Gamma \Rightarrow A}}{\Gamma \Rightarrow A}$ then $f_{\mathcal{D}}(\vec{x};\vec{y}) := \mathsf{s}_1(;f_{\mathcal{D}_0}(\vec{x};\vec{y}))$.

- If $\mathcal{D}$ is $\mathsf{cut}_N \dfrac{\overset{\mathcal{D}_0}{\Gamma \Rightarrow N} \quad \overset{\mathcal{D}_1}{\Gamma, N \Rightarrow A}}{\Gamma \Rightarrow A}$ then $f_{\mathcal{D}}(\vec{x};\vec{y}) := f_{\mathcal{D}_1}(\vec{x};\vec{y}, f_{\mathcal{D}_0}(\vec{x};\vec{y}))$.

- If $\mathcal{D}$ is $\mathsf{cut}_{\Box} \dfrac{\overset{\mathcal{D}_0}{\Gamma \Rightarrow \Box N} \quad \overset{\mathcal{D}_1}{\Box N, \Gamma \Rightarrow A}}{\Gamma \Rightarrow A}$ then $f_{\mathcal{D}}(\vec{x};\vec{y}) := f_{\mathcal{D}_1}(f_{\mathcal{D}_0}(\vec{x};\vec{y}), \vec{x};\vec{y})$.

- If $\mathcal{D}$ is $\mathsf{cond}_N \dfrac{\overset{\mathcal{D}_0}{\Gamma \Rightarrow N} \quad \overset{\mathcal{D}_1}{\Gamma, N \Rightarrow N} \quad \overset{\mathcal{D}_2}{\Gamma, N \Rightarrow N}}{\Gamma, N \Rightarrow N}$ then:

$$\begin{array}{rcll} f_{\mathcal{D}}(\vec{x};\vec{y},0) & := & f_{\mathcal{D}_0}(\vec{x};\vec{y}) & \\ f_{\mathcal{D}}(\vec{x};\vec{y},\mathsf{s}_0 y) & := & f_{\mathcal{D}_1}(\vec{x};\vec{y},y) & \text{if } y \neq 0 \\ f_{\mathcal{D}}(\vec{x};\vec{y},\mathsf{s}_1 y) & := & f_{\mathcal{D}_2}(\vec{x};\vec{y},y) & \end{array}$$

- If $\mathcal{D}$ is $\mathsf{cond}_{\Box} \dfrac{\overset{\mathcal{D}_0}{\Gamma \Rightarrow N} \quad \overset{\mathcal{D}_1}{\Box N, \Gamma \Rightarrow N} \quad \overset{\mathcal{D}_2}{\Box N, \Gamma \Rightarrow N}}{\Box N, \Gamma \Rightarrow N}$ then:

$$\begin{array}{rcll} f_{\mathcal{D}}(0,\vec{x};\vec{y}) & := & f_{\mathcal{D}_0}(\vec{x};\vec{y}) & \\ f_{\mathcal{D}}(\mathsf{s}_0 x,\vec{x};\vec{y}) & := & f_{\mathcal{D}_1}(x,\vec{x};\vec{y}) & \text{if } x \neq 0 \\ f_{\mathcal{D}}(\mathsf{s}_1 x,\vec{x};\vec{y}) & := & f_{\mathcal{D}_2}(x,\vec{x};\vec{y}) & \end{array}$$

- If $\mathcal{D}$ is $\mathsf{srec}_{\Box} \dfrac{\overset{\mathcal{D}_0}{\Gamma \Rightarrow N} \quad \overset{\mathcal{D}_1}{\Box N, \Gamma, N \Rightarrow N} \quad \overset{\mathcal{D}_2}{\Box N, \Gamma, N \Rightarrow N}}{\Box N, \Gamma \Rightarrow N}$ then:

$$\begin{array}{rcll} f_{\mathcal{D}}(0,\vec{x};\vec{y}) & := & f_{\mathcal{D}_0}(\vec{x};\vec{y}) & \\ f_{\mathcal{D}}(\mathsf{s}_0 x,\vec{x};\vec{y}) & := & f_{\mathcal{D}_1}(x,\vec{x};\vec{y}, f_{\mathcal{D}}(x,\vec{x};\vec{y})) & \text{if } x \neq 0 \\ f_{\mathcal{D}}(\mathsf{s}_1 x,\vec{x};\vec{y}) & := & f_{\mathcal{D}_2}(x,\vec{x};\vec{y}, f_{\mathcal{D}}(x,\vec{x};\vec{y})) & \end{array}$$

The above formal semantics exposes how B-derivations and B functions relate. The rule srec in Figure 2 corresponds to safe recursion, and safe composition along safe parameters is expressed by means of the rules cut_N. Note, however, that the function $f_{\mathcal{D}}$ is not quite defined according to function algebra B, due to the interpretation of the $\mathsf{cut}_{\Box}$ rule apparently not satisfying the required constraint on safe composition along a normal parameter. However, this admission turns out to be harmless, as exposited in the following proposition:

Proposition 5. *Given a* B*-derivation* $\mathcal{D} : \Box\Gamma, \vec{N} \Rightarrow \Box N$*, there is a* B*-derivation* $\mathcal{D}^* : \Box\Gamma \Rightarrow \Box N$ *(of smaller size) such that:*

$$f_{\mathcal{D}}(\vec{x};\vec{y}) = f_{\mathcal{D}^*}(\vec{x};).$$

Proof sketch. The proof is by induction on $\mathcal{D}$, using the fact that cond_N, $\mathsf{cond}_{\Box N}$ and $\mathsf{srec}_{\Box N}$ have only non-modal succedents. □

Our overuse of the notation B (for a function algebra and for a type system) is now justified by the following (folklore) result.

Proposition 6. *$f \in \mathsf{B}$ iff there is a B-derivation $\mathcal{D}$ for which $f_{\mathcal{D}}(\vec{x};\vec{y}) = f(\vec{x};\vec{y})$.*

Proof sketch. The left-right implication is routine and we omit it. For the right-left implication we proceed by induction on the structure of $\mathcal{D}$. In particular, notice that the interpretation $f_{\mathcal{D}}$ of a derivation $\mathcal{D}$ given in Definition 4 almost induces a bona fide interpretation of typing derivations into the function algebra, but for the case where the last rule of $\mathcal{D}$ is $\mathsf{cut}_{\Box}$. To handle this we appeal to Proposition 5, which allows us to infer that, if $\mathcal{D}_0 : \Box\Gamma, \vec{N} \Rightarrow \Box N$ is the sub-derivation of the leftmost premise of $\mathsf{cut}_{\Box}$, it can be replaced by the (smaller) subderivation $\mathcal{D}_0^* : \Box\Gamma \Rightarrow \Box N$ constructed in Proposition 5. From here, the interpretation from Definition 4 induces a correct instance of safe composition along a normal parameter. □

Convention 7. Given Proposition 6 above, we shall henceforth freely write $f(\vec{x};\vec{y}) \in \mathsf{B}$ if there is a derivation $\mathcal{D} : \Box\Gamma, \vec{N} \Rightarrow N$ with $f_{\mathcal{D}}(\vec{x};\vec{y}) = f(\vec{x};\vec{y})$.

3. Two-tiered circular systems on notation

In this section we introduce a 'coinductive' version of B, and we study global criteria that tame its computational strength. This proof-theoretic investigation will lead us to two relevant circular systems: CNB, which morally permits 'nested' versions of safe recursion, and CB, which will turn out to be closer to usual safe recursion.

Throughout this section we shall work with the set of typing rules $\mathsf{B}^- := \mathsf{B} \setminus \mathsf{srec}$.

3.1. Non-wellfounded typing derivations.

To begin with, we define the notion of 'coderivation', which is the fundamental formal object of this section.

Definition 8 (Coderivations)**.** A (B^--)*coderivation* is a possibly infinite rooted tree (of height $\leq \omega$) generated by the rules of B^-. Formally, it is a ternary tree $\mathcal{D} \subseteq \{0,1,2\}^*$ where each node of $\mathcal{D}$ is labelled by an inference step from B^- such that, whenever $\nu \in \mathcal{D}$ is labelled by a step $\dfrac{S_1 \quad \cdots \quad S_n}{S}$, for $n \leq 3$, ν has n children in $\mathcal{D}$ labelled by steps with conclusions $S_1, \ldots, S_n$ respectively. Sub-coderivations of a coderivation $\mathcal{D}$ rooted at position $\nu \in \{0,1,2\}^*$ will be denoted $\mathcal{D}_\nu$, so that $\mathcal{D}_\epsilon = \mathcal{D}$. We write $\nu \sqsubseteq \mu$ (resp. $\nu \sqsubset \mu$) if ν is a prefix (resp. a strict prefix) of μ, and in this case we say that μ is *above* (resp. *strictly above*) ν or that ν is *below* (resp. *strictly below*) μ. We extend this order from nodes to sequents in the obvious way.

We say that a coderivation is *regular* (or *circular*) if it has only finitely many distinct sub-coderivations.

If we ignore modalities, the rules of B^- can be obtained by adapting to binary notation the rules of CT_0, the type level 0 fragment of CT from [Das21], which are in turn derivable in the type level 0 fragment of C from [KPP21a]. (See Theorem 22 for further discussion on this point.)

Note that, while usual derivations may be naturally written as finite trees or dags, regular coderivations may be naturally written as finite directed (possibly cyclic) graphs:

Convention 9 (Representing coderivations)**.** Henceforth, we may mark steps by $\bullet$ (or similar) in a regular coderivation to indicate roots of identical sub-coderivations. Moreover, to avoid ambiguities and to ease parsing of (co)derivations, we shall often underline principal formulas of a rule instance in a given coderivation and omit instances of $\mathsf{w}_\Box$ and w_N as well as certain structural steps, e.g. during a cut step.

Example 10 (Regular coderivations)**.** The coderivations $\mathcal{I}$, $\mathcal{S}$ and $\mathcal{C}$ below are regular:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{N \Rightarrow N}\,\mathsf{id}}{N \Rightarrow N}\,\mathsf{s}_1}{\Box N \Rightarrow N}\,\Box_l}{\Box N \Rightarrow \Box N}\,\Box_r \qquad \dfrac{\vdots}{\Box N \Rightarrow N}\,\bullet}{\Box N \Rightarrow N}\,\mathsf{cut}_\Box\ \bullet$$

$$\dfrac{\dfrac{\dfrac{}{\Rightarrow N}\,0 \qquad \dfrac{}{N \Rightarrow N}\,\mathsf{id}}{\Rightarrow N}\,\mathsf{cut}_N \qquad \dfrac{\dfrac{}{N \Rightarrow N}\,\mathsf{id}}{\Box N \Rightarrow N}\,\Box_l}{\Box N \Rightarrow N}\,\mathsf{cond}_\Box \qquad \dfrac{\dfrac{\vdots}{\underline{\Box N} \Rightarrow N}\,\bullet \qquad \dfrac{}{N \Rightarrow N}\,\mathsf{id}}{\Box N \Rightarrow N}\,\mathsf{cut}_N}{\Box N \Rightarrow N}\,\mathsf{cond}_\Box\ \bullet$$

$$\dfrac{\dfrac{\dfrac{}{N \Rightarrow N}\,\mathsf{id}}{\underline{\Box N}, N \Rightarrow N}\,\mathsf{cond}_\Box \qquad \dfrac{\vdots}{\Box N, N \Rightarrow N}\,\mathsf{s}_i\ i{=}0{,}1}{\underline{\Box N}, N \Rightarrow N}\,\mathsf{cond}_\Box\ \circ \qquad \dfrac{\vdots}{\Box N, \Box N, N \Rightarrow N}\,\mathsf{cond}_\Box\ \bullet}{\underline{\Box N}, \Box N, N \Rightarrow N}\,\mathsf{s}_i\ i{=}0{,}1$$
$$\underline{\Box N}, \Box N, N \Rightarrow N\ \bullet\ \mathsf{cond}_\Box$$

As discussed in [Das21], and implicitly stated in [KPP21a], coderivations can be identified with Kleene-Herbrand-Gödel style equational programs, in general computing partial recursive functionals (see, e.g., [Kle71, §63] for further details). We shall specialise this idea to our two-sorted setting.

Definition 11 (Semantics of coderivations)**.** To each B^--coderivation $\mathcal{D}$ we associate a 'two-sorted' Kleene-Herbrand-Gödel partial function $f_\mathcal{D}$ obtained by construing the semantics of Definition 4 as a (possibly infinite) equational program.

Given a two-sorted function $f(\vec{x};\vec{y})$, we say that f is *defined* by a B^--coderivation $\mathcal{D}$ (or even, is B^-*-codefinable*) if $f_\mathcal{D}(\vec{x};\vec{y}) = f(\vec{x};\vec{y})$.

Note, in particular, that from a regular coderivation $\mathcal{D}$ we obtain a *finite* equational program determining $f_\mathcal{D}$. Of course, our overuse of the notation $f_\mathcal{D}$ is suggestive since it is consistent with that of Definition 4.

Example 12 (Regular coderivations, revisited)**.** Let us consider the semantics of the coderivations $\mathcal{I}$, $\mathcal{S}$ and $\mathcal{C}$ from Example 10.

- The partial functions $f_{\mathcal{I}\nu}$ are given by the following equational program:

$$\begin{array}{rcl} f_{\mathcal{I}_\epsilon}(x;) & = & f_{\mathcal{I}_1}(f_{\mathcal{I}_0}(x;);) \\ f_{\mathcal{I}_0}(x;) & = & f_{\mathcal{I}_{00}}(x;) \\ f_{\mathcal{I}_{00}}(x;) & = & f_{\mathcal{I}_{000}}(;x) \\ f_{\mathcal{I}_{000}}(;x) & = & \mathsf{s}_1(;x) \\ f_{\mathcal{I}_1}(x;) & = & f_{\mathcal{I}_\epsilon}(x;) \end{array}$$

  By purely equational reasoning, we can simplify this program to obtain:

$$f_{\mathcal{I}_\epsilon}(x;) \quad = \quad f_{\mathcal{I}_\epsilon}(\mathsf{s}_1 x;)$$

  Since the above equational program keeps increasing the input, the function $f_{\mathcal{I}} = f_{\mathcal{I}_\epsilon}$ is always undefined.
- The equational program associated with the coderivation $\mathcal{S}$ is as follows:

$$\begin{array}{rcl} f_{\mathcal{S}_\epsilon}(0;) & = & f_{\mathcal{S}_0}(;) \\ f_{\mathcal{S}_\epsilon}(\mathsf{s}_0 x;) & = & f_{\mathcal{S}_1}(x;) \\ f_{\mathcal{S}_\epsilon}(\mathsf{s}_1 x;) & = & f_{\mathcal{S}_2}(x;) \\ f_{\mathcal{S}_0}(;) & = & \mathsf{s}_1 0 \\ f_{\mathcal{S}_1}(x;) & = & \mathsf{s}_1 x \\ f_{\mathcal{S}_2}(x;) & = & f_{\mathcal{S}_{21}}(f_{\mathcal{S}_{20}}(x;);) \\ f_{\mathcal{S}_{20}}(x;) & = & f_{\mathcal{S}_\epsilon}(x;) \\ f_{\mathcal{S}_{21}}(x;) & = & f_{\mathcal{S}_{210}}(;x) \\ f_{\mathcal{S}_{210}}(;x) & = & \mathsf{s}_0(;x) \end{array}$$

  By equational reasoning again we obtain:

$$\begin{array}{rcll} f_{\mathcal{S}_\epsilon}(0;) & = & \mathsf{s}_1 0 & \\ f_{\mathcal{S}_\epsilon}(\mathsf{s}_0 x;) & = & \mathsf{s}_1 x & x \neq 0 \\ f_{\mathcal{S}_\epsilon}(\mathsf{s}_1 x;) & = & \mathsf{s}_0 f_{\mathcal{S}_\epsilon}(x;) & \end{array}$$

  Note, therefore, that the function $f_{\mathcal{S}} = f_{\mathcal{S}_\epsilon}$ computes the unary successor function $x \mapsto x + 1$.
- Last, the equational program of $\mathcal{C}$ can be written as:

$$\begin{array}{rcll} f_{\mathcal{C}_\epsilon}(0,0;z) & = & z & \\ f_{\mathcal{C}_\epsilon}(0,\mathsf{s}_i y;z) & = & \mathsf{s}_i f_{\mathcal{C}_\epsilon}(x,y;z) & \neq 0 \\ f_{\mathcal{C}_\epsilon}(\mathsf{s}_i x,y;z) & = & \mathsf{s}_i f_{\mathcal{C}_\epsilon}(x,y;z) & \neq 0 \end{array}$$

  which computes concatenation of the binary representation of three natural numbers.

Despite regular coderivations being finitely presentable, they may introduce functions that are undefined for some arguments, as shown in the above example. We shall adapt to our setting a well-known 'termination criterion' from non-wellfounded proof theory able to identify certain coderivations (even the non-regular ones) that define total functions. First, let us recall some standard proof theoretic concepts about (co)derivations, similar to those occurring in [Das21, KPP21b].

Definition 13 (Ancestry)**.** Fix a coderivation $\mathcal{D}$. We say that a type occurrence A is an *immediate ancestor* of a type occurrence B in $\mathcal{D}$ if they are types in a premiss and conclusion (respectively) of an inference step and, as typeset in Figure 2, have the same colour. If A and B are in some Γ or Γ', then we furthermore insist that they are in the same position in the list.

Being a binary relation, immediate ancestry forms a directed graph upon which our correctness criterion is built.

Definition 14 (Progressing coderivations)**.** A *thread* is a maximal path in the graph of immediate ancestry. We say that a (infinite) thread is *progressing* if it is eventually constant $\Box N$ and infinitely often principal for a $\mathsf{cond}_{\Box N}$ rule.

A coderivation is *progressing* if each of its infinite branches is progressing.

Example 15 (Regular coderivations re-revisited)**.** In Example 10, both $\mathcal{I}$ and $\mathcal{S}$ have precisely one infinite branch (that loops on $\bullet$). The former is not progressing because the infinite branch contains no instances of $\mathsf{cond}_{\Box}$ at all. By contrast, $\mathcal{S}$ is progressing, since the infinite branch has a progressing thread on the blue formulas $\Box N$ indicated there. Last, $\mathcal{C}$ has two simple loops, one on $\bullet$ and the other one on $\circ$. For any infinite branch B we have two cases:

- if B crosses the bottommost conditional infinitely many times, it contains a progressing blue thread;
- otherwise, B crosses the topmost conditional infinitely many times, so that it contains a progressing red thread.

Therefore, $\mathcal{C}$ is progressing.

As shown in previous works (see e.g. [Das21] and [KPP21a]), the progressing criterion is sufficient to guarantee that the partial function computed is, in fact, a well-defined total function:

Proposition 16 (Totality)**.** *If $\mathcal{D}$ is progressing then $f_{\mathcal{D}}$ is total.*

Proof sketch. We proceed by contradiction. If $f_{\mathcal{D}}$ is non-total then, since each rule preserves totality, we must have that $f_{\mathcal{D}'}$ is non-total for one of $\mathcal{D}$'s immediate sub-coderivations $\mathcal{D}'$. Continuing this reasoning we can build an infinite 'non-total' branch $B = (\mathcal{D}^i)_{i<\omega}$. Let $(\Box N^i)_{i\geq k}$ be a progressing thread along B, and assign to each $\Box N^i$ the least natural number $n_i \in \mathbb{N}$ such that $f_{\mathcal{D}^i}$ is non-total when n_i is assigned to the type occurrence $\Box N^i$.

Now, notice that:

- $(n_i)_{i\geq k}$ is monotone non-increasing, by inspection of the rules and their interpretations from Definition 4.
- $(n_i)_{i\geq k}$ does not converge, since $(\Box N^i)_{i\geq k}$ is progressing and so is infinitely often principal for $\mathsf{cond}_{\Box}$, where the value of n_i must strictly decrease (cf., again, Definition 4).

This contradicts the well-ordering property of the natural numbers. □

One of the most appealing features of the progressing criterion is that, while being rather expressive and admitting many natural programs, e.g. as we will see in the next subsections, it remains effective (for regular coderivations) thanks to well known arguments in automaton theory:

Fact 17 (Folklore)**.** *It is decidable whether a regular coderivation is progressing.*

This well-known result (see, e.g., [DHL06a] for an exposition for a similar circular system) follows from the fact that the progressing criterion is equivalent to the universality of a Büchi automaton of size determined by the (finite) representation of the input coderivation. This problem is decidable in polynomial space, though

the correctness of this algorithm requires nontrivial infinitary combinatorics, as formally demonstrated in [KMPS19].

Let us finally observe that the progressiveness condition turns out to be sufficient to restate Proposition 5 in a non-wellfounded setting:

Proposition 18. *Given a progressing* B^-*-coderivation* $\mathcal{D} : \Box\Gamma, \vec{N} \Rightarrow \Box N$*, there is a progressing* B^-*-coderivation* $\mathcal{D}^* : \Box\Gamma \Rightarrow \Box N$ *such that:*

$$f_{\mathcal{D}}(\vec{x};\vec{y}) = f_{\mathcal{D}^*}(\vec{x};).$$

Proof. By progressiveness, any infinite branch contains a $\mathsf{cond}_\Box$-step, which has non-modal succedent. Since the conclusion has a modal succedent, by inspection on the rules of B^- we conclude that each infinite branch contains an instance of $\Box_r$. Hence, the set of the conclusions of such $\Box_r$-steps forms a bar across $\mathcal{D}$. By applying weak König Lemma, the set of all nodes of $\mathcal{D}$ below this bar, say $X_{\mathcal{D}}$, is finite. The proof is by induction on the cardinality of $X_{\mathcal{D}}$ and is analogous to Proposition 5. □

Note that the above proof uniquely depends on progressiveness, and so it holds for non-regular progressing B^--coderivations as well.

3.2. **Computational strength of coderivations.** The sequent calculus system B (Figure 2) is a $S4$-style modal calculus transplanting Bellantoni and Cook's function algebra into a proof-theoretical setting (as in [Hof97]). In particular, B-derivations locally introduce safe recursion by means of the rule srec, and Proposition 5 ensures that the composition schemes defined by the cut rules are safe. A different situation arises when we move from B-derivations to B^--coderivations: non-wellfoundedness can actually be used to circumvent the normal/safe distinction and subsume more complex recursion mechanisms. The following example illustrates the problem.

Example 19 (Non-safety in regular progressing coderivations)**.** Let $g(\vec{x};\vec{y})$ and $h_i(x,\vec{x},z;\vec{y})$ with $i = 0,1$ be functions definable by some (regular and progressing) B^--coderivations $\mathcal{G}$ and $\mathcal{H}_i$, respectively. Then the following *non-safe* recursion scheme,

$$\begin{array}{rcl} f(0,\vec{x};) & = & g(\vec{x};) \\ f(\mathsf{s}_i x,\vec{x};) & = & h_i(x,\vec{x},f(x,\vec{x};);) \end{array} \tag{1}$$

can be defined by the following (regular and progressing) B^--coderivation:

$$\dfrac{\dfrac{\dfrac{\vdots}{\mathsf{cond}_\Box\dfrac{\Box N,\Box\vec{N}\Rightarrow N}{\bullet}}}{\Box_r\dfrac{\Box N,\Box\vec{N}\Rightarrow \Box N}{}} \quad \mathcal{G}\ \Box\vec{N}\Rightarrow N}{\mathsf{cond}_\Box\dfrac{\mathsf{cut}_\Box\dfrac{}{\Box N,\Box\vec{N}\Rightarrow N}\ i=0,1}{\underline{\Box N},\Box\vec{N}\Rightarrow N}\bullet} \qquad \Box N,\Box\vec{N},\Box N\Rightarrow N \ \ \mathcal{H}_i$$

Note in particular that we can use the scheme in Equation (1) to define all primitive recursive functions.

We can go further and show that, without restrictions on progressing coderivations, the distinction between modal and non-modal inputs becomes redundant:

Proposition 20. *For any coderivation $\mathcal{D} : \Box N, \overset{n}{\ldots}, \Box N, N, \overset{m}{\ldots}, N \Rightarrow N$ of B^- there is a coderivation $\mathcal{D}^\Box : \Box N, \overset{n+m}{\ldots}, \Box N \Rightarrow N$ such that:*

- *$f_{\mathcal{D}}(\vec{x};\vec{y}) = f_{\mathcal{D}^\Box}(\vec{x},\vec{y};)$;*
- *$\mathcal{D}^\Box$ does not contain instances of w_N, e_N, cut_N, cond_N.*

Moreover, $\mathcal{D}^\Box$ is regular (resp. progressing) if $\mathcal{D}$ is.

Proof. We construct $\mathcal{D}^\Box$ coinductively. The only interesting cases are when $\mathcal{D}$ is id or it is obtained from $\mathcal{D}_0 : \Gamma \Rightarrow N$ and $\mathcal{D}_1 : \Gamma, N \Rightarrow A$ by applying a cut_N-step. Then, $\mathcal{D}^\Box$ is constructed, respectively, as follows:

$$\dfrac{}{N \Rightarrow N}\,\mathsf{id} \qquad \dfrac{}{\Box N \Rightarrow N}\,\Box_l \qquad\qquad \dfrac{\dfrac{\overset{\mathcal{D}_0^\Box}{\Box\Gamma \Rightarrow N}}{\Box\Gamma \Rightarrow \Box N}\,\Box_r \qquad \dfrac{\overset{\mathcal{D}_1^\Box}{\Box\Gamma, N \Rightarrow N}}{\Box\Gamma, \Box N \Rightarrow N}\,\Box_l}{\Box\Gamma \Rightarrow N}\,\mathsf{cut}_{\Box N} \qquad \Box$$

As we shall see in the next subsections, the above issues can be avoided by imposing a further global criterion to recover safe recursion in a non-wellfounded setting. On the other hand, Proposition 20 turns out to be particularly useful to illustrate how regularity and progressiveness affect the class of B^--definable functions. Indeed, thanks to the above proposition, we can recast several of the computational results present in the (type 0) systems from [Das21, KPP21b]:

Proposition 21. *We have the following:*

(1) If $f(\vec{x})$ is primitive recursive then $f(\vec{x};)$ is defined by a regular and progressing B^--coderivation.
(2) The class of regular B^--coderivations is Turing-complete, i.e. they define every partial recursive function.
(3) Any function $f(\vec{x};)$ is defined by a progressing B^--coderivation.

Self-contained demonstrations of these points can be found in Appendix B.1.

In light of the proposition above, it is natural to wonder at this point precisely what class of functions is defined by the regular progressing B^--coderivations. Interestingly [KPP21b] show that, in the absence of contraction rules, *only* the primitive recursive funtions are so definable (even at all finite types). It is tempting, therefore, to conjecture that the regular and progressing B^--coderivations define *just* the primitive recursive functions.

However there is a crucial difference between our formulation of cut and that in [KPP21b], namely that ours is context-sharing and theirs is context-splitting. Thus the former admits a quite controlled form of contraction that, perhaps surprisingly, is enough to simulate the type 0 fragment CT_0 from [Das21] (which has explicit contraction):

Theorem 22. *$f(\vec{x};)$ is defined by a regular progressing B^--coderivation if and only if $f(\vec{x})$ is type-1-primitive-recursive, i.e. it is in the fragment T_1 of Gödel's T with recursion only at type 1.*

The proof of this result is somewhat technical, relying on some results from [Das21], and is somewhat orthogonal to the focus of this paper, so is devolved to Appendix B.2.

Note however that, given the computationally equivalent system CT_0 with contraction from [Das21], we can view the above result as a sort of 'contraction admissibility' for regular progressing B^--coderivations. Let us take a moment to make this formal.

Call $\mathsf{B}^- + \{\mathsf{c}_N, \mathsf{c}_{\Box N}\}$ the extension of B^- with the rules c_N and $\mathsf{c}_{\Box N}$ below:

$$\mathsf{c}_N\ \frac{\Gamma, N, N \Rightarrow B}{\Gamma, N \Rightarrow B} \qquad \mathsf{c}_\Box\ \frac{\Gamma, \Box N, \Box N \Rightarrow B}{\Gamma, \Box N \Rightarrow B} \tag{2}$$

where the semantics for the new system extends the one for B^- in the obvious way, and the notion of (progressing) thread is induced by the given colouring.[2] We have:

Corollary 23. *$f(\vec{x};)$ is definable by a regular progressing $\mathsf{B}^- + \{\mathsf{c}_N, \mathsf{c}_{\Box N}\}$-coderivation iff it is definable by a regular progressing B^--coderivation.*

Proof idea. The former system is equivalent to CT_0 from [Das21], whose type 1 functions are just those of T_1 by [Das21, Corollary 80], which are all defined by regular progressing B^--coderivations by Theorem 22 above. □

Remark 24. The reader may at this point wonder if a direct 'contraction-admissibility' argument exists for the rules in (2). First, notice that c_N can be derived in B^-:

$$\mathsf{cut}_N\ \frac{\mathsf{w}_N\ \dfrac{\mathsf{id}\ \dfrac{}{N \Rightarrow N}}{\Gamma, N \Rightarrow N} \qquad \Gamma, N, N \Rightarrow B}{\Gamma, N \Rightarrow B}$$

While a similar derivation exists for $\mathsf{c}_\Box$, note that this crucially does not preserve the same notion of thread (cf. colours above) and so does not, a priori, preserve progressiveness.

3.3. **Proof-level conditions motivated by implicit complexity.** Example 19 from the previous subsection motivates the introduction of a further criterion on B^--coderivations to rule out recursion schemes that are 'not safe'. To this end we propose the following simple criterion:

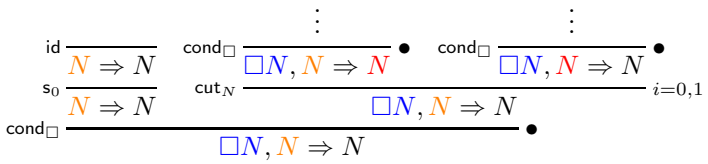
Definition 25 (Safety)**.** A B^--coderivation is *safe* if each infinite branch crosses only finitely many $\mathsf{cut}_\Box$-steps.

Example 26. The coderivation $\mathcal{I}$ in Example 10 is not safe, as there is an instance of $\mathsf{cut}_\Box$ in the loop on $\bullet$, which means that there is an infinite branch crossing infinitely many $\mathsf{cut}_\Box$-steps. By contrast, using the same reasoning, we can infer that the coderivations $\mathcal{S}$ and $\mathcal{C}$ are safe.

By inspecting the coderivation of Example 19, we notice that the infinite branch that loops on $\bullet$ contains infinitely many $\mathsf{cut}_\Box$ steps, so that it does not respect the above safety condition. Perhaps surprisingly, however, the safety condition is not enough to restrict the set of B^--definable functions to the polytime computable ones, as the following example shows.

[2] Note that the totality argument of Proposition 16 still applies in the presence of these rules, cf. also [Das21, KPP21b].

Example 27 (Safe exponentiation)**.** Consider the following coderivation $\mathcal{E}$,



where we identify the sub-coderivations above the second and the third premises of the conditional. The coderivation is clearly progressing. Moreover it is safe, as $\mathcal{E}$ has no instances of $\mathsf{cut}_\Box$. Its associated equational program can be written as follows:

$$\begin{array}{rcll} f_{\mathcal{E}}(0;y) & = & \mathsf{s}_0(;y) & \\ f_{\mathcal{E}}(\mathsf{s}_0 x;y) & = & f_{\mathcal{E}}(x;f_{\mathcal{E}}(x;y)) & x \neq 0 \\ f_{\mathcal{E}}(\mathsf{s}_1 x;y) & = & f_{\mathcal{E}}(x;f_{\mathcal{E}}(x;y)) & \end{array} \tag{3}$$

The above function $f_{\mathcal{E}}$ has already appeared in [Hof97, Lei99]. It is not hard to show, by induction on x, that $f_{\mathcal{E}}(x;y) = 2^{2^{|x|}} \cdot |y|$. Thus $f_{\mathcal{E}}$ has exponential growth rate (as long as $y \neq 0$), despite the program defining a 'safe' recursion scheme.

The above coderivation exemplifies a safe recursion scheme that is able to *nest* one recursive call inside another in order to obtain exponential growth rate. This is in fact a peculiar feature of regular proofs, and it is worth discussing.

As we have already seen, namely in Theorem 22, (progressing) B^--coderivations are able to simulate some sort of higher-order recursion, namely at type 1 (cf. also [Das21]). In this way it is not surprising that the sort of 'nested recursion' in Equation (3) is definable since type 1 recursion, in particular, allows such nesting of the recursive calls. To make this point more apparent, consider the following higher-order recursion operator:

$$\mathsf{rec}_A : \Box N \to (\Box N \to A \to A) \to A \to A$$

with $A = N \to N$, and $f(x) = \mathsf{rec}_A(x,h,g)$ is defined as $f(0) = g$ and $f(\mathsf{s}_i x) = h(x, f(x))$ for $x > 0$. By setting $g := \lambda y : N.\mathsf{s}_0 y$ and $h := \lambda x : \Box N.\lambda u : N \to N.(\lambda y : N.u(u\, y))$ we can easily check that $f_{\mathcal{E}}(x;y) = \mathsf{rec}_A(x,h,g)(y)$, where $\mathcal{E}$ is as in Example 27. Hence, the function $f_{\mathcal{E}}(x;y)$ can be defined by means of a higher-order version of safe recursion.

As noticed by Hofmann [Hof97], and formally proved by Leivant [Lei99], thanks to the capability of nesting recursive calls, higher-order safe recursion can be used to characterise **FELEMENTARY**. In particular, Hofmann showed in [Hof97] that by introducing a linearity restriction to the operator rec_A, which prevents duplication of recursive calls, it is possible to recover the class **FPTIME**. The resulting type system, called SLR ('Safe Linear Recursion'), can be then regarded as a higher-order formulation of B.

Following [Hof97], we shall impose a linearity criterion to rule out those coderivations that nest recursive calls. This is achieved by observing that the duplication of the recursive calls of $f_{\mathcal{E}}$ in Example 27 is due to the presence in $\mathcal{E}$ of loops on $\bullet$ crossing both premises of a cut_N step. Hence, our circular-proof-theoretic counterpart of Hofmann's linearity restriction can be obtained by demanding that at most one premise of each cut_N step is crossed by such loops. We shall actually present a slightly more general criterion which does not depend on our intuitive notion of loop.

Definition 28 (Left-leaning)**.** A B^--coderivation is *left-leaning* if each infinite branch goes right at a cut_N-step only finitely often.

Example 29. In Example 10, $\mathcal{I}$ is trivially left-leaning, as it contains no instances of cut_N at all. The coderivations $\mathcal{S}$ and $\mathcal{C}$ are also left-leaning, since no infinite branch can go right at the cut_N steps. By contrast, the coderivation $\mathcal{E}$ in Example 27 is not left-leaning, as there is an infinite branch looping at $\bullet$ and crossing infinitely many times the rightmost premise of the cut_N-step.

We are now ready to define our circular systems:

Definition 30 (Circular Implicit Systems)**.** We define CNB as the class of safe progressing regular B^--coderivations. CB is the restriction of CNB to only left-leaning coderivations. A function $f(\vec{x};\vec{y})$ is CNB-*definable* (or CB-*definable*) if there is a coderivation $\mathcal{D} \in \mathsf{CNB}$ (resp., $\mathcal{D} \in \mathsf{CB}$) such that $f_{\mathcal{D}}(\vec{x};\vec{y}) = f(\vec{x};\vec{y})$.

Let us point out that Proposition 18 can be stated to preserve safety and left-leaningness:

Proposition 31. *Given $\mathcal{D} : \Box\Gamma, \vec{N} \Rightarrow \Box N$ a coderivation in CNB (or CB), there exists CNB-coderivation (resp., CB-coderivation) $\mathcal{D}^* : \Box\Gamma \Rightarrow \Box N$ such that:*

$$f_{\mathcal{D}}(\vec{x};\vec{y}) = f_{\mathcal{D}^*}(\vec{x};)$$

3.4. **On the complexity of proof-checking.** Note that both the safety and the left-leaning conditions above are defined at the level of arbitrary coderivations, not just regular and/or progressing ones. Moreover, these conditions are easy to check on regular coderivations:

Proposition 32. *The safety and the left-leaning condition are* **NL***-decidable for regular coderivations.*

Proof. We can represent a regular coderivation $\mathcal{D}$ as a finite directed (possibly cyclic) graph $G_{\mathcal{D}}$ labelled with inference rules. Then, the problem of deciding whether $\mathcal{D}$ is not safe (resp. left leaning) comes down to the problem of deciding whether a cycle π of $G_{\mathcal{D}}$ exists crossing a node labelled $\mathsf{cut}_\Box$ (resp. crossing the rightmost child of a node labelled cut_N). W.l.o.g. we can assume that π is a simple cycle. Given (an encoding of) $G_{\mathcal{D}}$ as read-only input and (an encoding of) π as a read-only certificate, we can easily construct a deterministic Turing machine M verifying that the cycle π crosses $\mathsf{cut}_\Box$ (resp. the rightmost child of a node cut_N) in $G_{\mathcal{D}}$. More specifically:

- the size of the certificate is smaller than the size of the description of $G_{\mathcal{D}}$;
- M reads once from left to right the addresses of the certificate which provide the information about where to move the pointers;
- M works in logspace as it only stores addresses in memory. □

Recall that progressiveness of regular coderivations is decidable by reduction to universality of Büchi automata, a **PSPACE**-complete problem. Indeed progressiveness itself is **PSPACE**-complete in many settings, cf. [NST19]. It is perhaps surprising, therefore, that the safety of a regular coderivation also allows us to decide progressiveness efficiently too, thanks to the following reduction:

Proposition 33. *A safe B^--coderivation is progressing iff every infinite branch has infinitely many $\mathsf{cond}_\Box$-steps.*

Proof. The left-right implication is trivial. For the right-left implication, let us consider an infinite branch B of a safe B^--coderivation $\mathcal{D}$. By safety, there exists a node ν of B such that any sequent above ν in B is not the conclusion of a $\mathsf{cut}_\Box$-step. Now, by inspecting the rules of $\mathsf{B}^- \setminus \{\mathsf{cut}_\Box\}$ we observe that:

- every modal formula occurrence in B has a unique thread along B;
- any infinite thread along B cannot start strictly above ν.

Hence, setting k to be the number of $\Box N$ occurrences in the antecedent of ν, B has (at most) k infinite threads. Moreover, since B contains infinitely many $\mathsf{cond}_\Box$-steps, by the Infinite Pigeonhole Principle we conclude that one of these threads is infinitely often principal for the $\mathsf{cond}_\Box$ rule. □

Thus, using similar reasoning to that of Proposition 32 we may conclude from Proposition 33 the following:

Proposition 34. *Given a regular B^--coderivation $\mathcal{D}$, the problem of deciding if $\mathcal{D}$ is in CNB (resp. CB) is in* **NL**.

Let us point out that the reduction above is similar to and indeed generalises an analogous one for cut-free coderivations for extensions of Kleene algebra, cf. [DP18, Proposition 8].

Remark 35. Recall that in Proposition 20 we showed by means of an indirect argument that explicit contraction is 'admissible' for regular progressing B^--coderivations, and in Remark 24 we pointed out that a more direct proof of this result cannot easily be found. In fact, we could not easily find such a direct contraction-admissibility even for CB or CNB. However, let us point out that Proposition 33 still holds when explicit contraction is added to B^-. On the one hand, this observation witnesses the robustness of the safety criterion; on the other hand, it stresses that, whatever class of functions CB and CNB define in the presence explicit contraction, the resulting circular proof systems are still **NL**-checkable (cf. Proposition 34). A detailed argument for this can be found in Appendix B.3.

4. Some variants of safe recursion on notation

In this section we shall introduce various extensions of B to classify the expressivity of the circular systems CB and CNB. First, starting from the analysis of Example 27 and the subsequent system CNB, we shall define a version of B with safe *nested* recursion, called NB. Second, motivated by the more liberal way of defining functions in both CB and CNB, we shall endow the function algebras B and NB with forms of safe recursion over a well-founded relation $\subset$ on lists of normal parameters. Figure 3 summarises the function algebras considered and their relations.

4.1. **Function algebras with nesting and composition during recursion.** One of the key features of safe recursion is that 'nesting' of recursive calls is not permitted. For instance, recalling the coderivation $\mathcal{E}$ from Example 27, let us revisit the following equational program:

$$\begin{array}{rcl} \mathsf{ex}(0;y) & = & \mathsf{s}_0 y \\ \mathsf{ex}(\mathsf{s}_i x;y) & = & \mathsf{ex}(x;\mathsf{ex}(x;y)) \end{array} \tag{4}$$

Recall that $\mathsf{ex}(x;y) = f_{\mathcal{E}}(x;y) = 2^{2^{|x|}} \cdot y$. The 'recursion step' on the second line is compatible with safe composition, in that safe inputs only occur in hereditarily

safe recursion	on notation	on $\subset$
unnested	B	$\mathsf{B}^{\subset}$
unnested, with compositions	SB	$\mathsf{SB}^{\subset}$
nested	NB	$\mathsf{NB}^{\subset}$

FIGURE 3. The function algebras considered in Section 4. Any algebra is included in one below it or to the right of it.

safe positions, but one of the recursive calls takes another recursive call among its safe inputs. In Example 27 we showed how the above function $\mathsf{ex}(x;y)$ can be CNB-defined. It is thus reasonable to look for a suitable extension of B able to formalise such nested recursion to serve as a function algebraic counterpart to CNB.

It will be convenient throughout this section to work with generalisations of B including *oracles*. Formally speaking, these can be seen as variables ranging over two-sorted functions, though we shall often identify them with functions themselves.

Definition 36. For all sets of oracles $\vec{a} = a_1, \ldots, a_k$, we define the algebra of B^- functions *over* $\vec{a}$ to include all the initial functions of B^- and,

- (oracles). $a_i(\vec{x};\vec{y})$ is a function over $\vec{a}$, for $1 \leq i \leq k$, (where $\vec{x}$,$\vec{y}$ have appropriate length for a_i).

and closed under:

- (Safe Composition).
  (1) from $g(\vec{x};\vec{y}), h(\vec{x};\vec{y},y)$ over $\vec{a}$ define $f(\vec{x};\vec{y})$ over $\vec{a}$ by $f(\vec{x};\vec{y}) = h(\vec{x};\vec{y},g(\vec{x};\vec{y}))$.
  (2) from $g(\vec{x};)$ over $\varnothing$ and $h(\vec{x},x;\vec{y})$ over $\varnothing$ define $f(\vec{x};\vec{y})$ over $\vec{a}$ by $f(\vec{x};\vec{y}) = h(\vec{x},g(\vec{x};);\vec{y})$.

We write $\mathsf{B}^-(\vec{a})$ for the class of functions over $\vec{a}$ generated in this way.

Note that Safe Composition along normal parameters (Item 2 above) comes with the condition that $g(\vec{x};)$ is oracle-free. This restriction prevents oracles (and hence the recursive calls) appearing in normal position. The same condition on $h(\vec{x},x;\vec{y})$ being oracle-free is not strictly necessary for the complexity bounds we are after, as we shall see in the next section when we define more expressive algebras, but is convenient in order to facilitate the 'grand tour' strategy of this paper (cf. Figure 1).

We shall write, say, $\lambda\vec{v}.f(\vec{x};\vec{v})$ for the function taking only safe arguments $\vec{v}$ with $(\lambda\vec{v}.f(\vec{x};\vec{v}))(;\vec{y}) = f(\vec{x};\vec{y})$. Nested recursion can be formalised in the setting of algebras-with-oracles as follows:

Definition 37 (Safe Nested Recursion)**.** We write snrec for the scheme:

- from $g(\vec{x};\vec{y})$ over $\varnothing$ and $h(a)(x,\vec{x};\vec{y})$ over $a,\vec{a}$, define $f(x,\vec{x};\vec{y})$ over $\vec{a}$ by:

$$\begin{aligned} f(0,\vec{x};\vec{y}) &= g(\vec{x};\vec{y}) \\ f(\mathsf{s}_0 x,\vec{x};\vec{y}) &= h_0(\lambda\vec{v}.f(x,\vec{x};\vec{v}))(x,\vec{x};\vec{y}) \quad x \neq 0 \\ f(\mathsf{s}_1 x,\vec{x};\vec{y}) &= h_1(\lambda\vec{v}.f(x,\vec{x};\vec{v}))(x,\vec{x};\vec{y}) \end{aligned}$$

We write $\mathsf{NB}(\vec{a})$ for the class of functions over $\vec{a}$ generated from B^- under snrec and Safe Composition (from Definition 36), and write simply NB for $\mathsf{NB}(\varnothing)$.

Note that safe nested recursion, per se, admits definitions that are not morally 'nested' but rather use a form of 'composition during recursion'. Such a scheme might have the form:

- from $g(\vec{x};\vec{y}), \vec{g}_1(x,\vec{x};\vec{y}), \ldots, \vec{g}_k(x,\vec{x};\vec{y})$ and $h_i(x,\vec{x};\vec{y},z_1,\ldots,z_k)$ define:

$$\begin{aligned} f(0,\vec{x};\vec{y}) &= g(\vec{x};\vec{y}) \\ f(\mathsf{s}_i x,\vec{x};\vec{y}) &= h_i(x,\vec{x};\vec{y},f(x,\vec{x};\vec{g}_1(x,\vec{x};\vec{y})),\ldots,f(x,\vec{x};\vec{g}_k(x,\vec{x};\vec{y}))) \end{aligned} \tag{5}$$

In this case, note that we have allowed the safe inputs of f to take arbitrary values given by previously defined functions, but at the same time f never calls itself in a safe position, as in (4). Note that this safe recursion scheme does not require the use of oracles in its statement, but is equivalent to the following one using a notion of 'nesting' for Safe Composition:

Definition 38 (Safe Recursion with Composition During Recursion)**.** An instance of Safe Composition along Safe Parameters (Item 1 from Definition 36) is called *unnested* if (at least) one of $g(\vec{x};\vec{y})$ and $h(\vec{x};\vec{y},y)$ is over $\varnothing$ (i.e. is oracle-free). We define $\mathsf{SB}(\vec{a})$ to be the restriction of $\mathsf{NB}(\vec{a})$ using only unnested Safe Composition along Safe Parameters, Item 1 from Definition 36, and write simply SB for $\mathsf{SB}(\varnothing)$.

It is not hard to show that, indeed, SB defines the same class of functions of the extension of B^- by the scheme given in eq. (5). On the one hand, given h_i and $\vec{g}_j$, the scheme in (5) can be defined in $\mathsf{NB}(\vec{a})$ using snrec and only unnested instances of safe composition. On the other hand, given $h(a)(x,\vec{x};\vec{y})$ over oracles $a,\vec{a}$, snrec allows us to define $f(\mathsf{s}_i x,\vec{x};\vec{y})$ by replacing the oracle a in h with the recursive calls $\lambda\vec{v}.f(x,\vec{x};\vec{v})$. However, since $h(a)$ is defined using unnested safe composition, no recursive call of f can be found along the safe parameters of another one.

It should be said that we will not actually use SB in this work, but rather an extension of it defined in the next section, but we have included it for completeness of our exposition.

4.2. **Safe recursion on well-founded relations.** Function algebras in general may be readily extended by recursion on arbitrary well-founded relations. For instance, given a well-founded preorder $\trianglelefteq$, and writing $\triangleleft$ for its strict variant,[3] 'safe recursion on $\triangleleft$' is given by the following scheme:

- from $h(a)(x,\vec{x};\vec{y})$ over $a,\vec{a}$, define $f(x,\vec{x};\vec{y})$ over $\vec{a}$ by:

$$f(x,\vec{x};\vec{y}) = h(\lambda v \triangleleft x.f(v,\vec{x};\vec{y}))(x,\vec{x};\vec{y})$$

Note here that we employ the notation $\lambda v \triangleleft x$ for a 'guarded abstraction'. Formally:

$$(\lambda v \triangleleft x.f(v,\vec{x};\vec{y}))(v) := \begin{cases} f(v,\vec{x};\vec{y}) & v \triangleleft y \\ 0 & \text{otherwise} \end{cases}$$

It is now not hard to see that total functions (with oracles) are closed under the recursion scheme above, by reduction to induction on the well-founded relation $\triangleleft$.

Note that such schemes can be naturally extended to preorders on *tuples* of numbers too, by abstracting several inputs. We shall specialise this idea to a particular well-founded preorder that will be helpful later to bound the complexity of definable functions in our systems CB and CNB.

[3]To be precise, in this work, for a preorder $\trianglelefteq$ we write $x \triangleleft y$ if $x \trianglelefteq y$ and $y \not\trianglelefteq x$. As an abuse of terminology, we say that $\trianglelefteq$ is well-founded just when $\triangleleft$ is.

Recall that we say that x is a *prefix* of y if y can be written $x2^n + z$ for some $n \geq 0$ and some $z < 2^n$, i.e. y has the form xz in binary notation. We say that x is a *strict* prefix of y if x is a prefix of y but $x \neq y$.

Definition 39 (Permutations of prefixes)**.** Let $[n] := \{0, \ldots, n-1\}$. We shall write $(x_0, \ldots, x_{n-1}) \subseteq (y_0, \ldots, y_{n-1})$ if, for some permutation $\pi : [n] \to [n]$, we have that x_i is a prefix of $y_{\pi i}$, for all $i < n$. We write $\vec{x} \subset \vec{y}$ if $\vec{x} \subseteq y$ but $\vec{y} \not\subseteq x$, i.e. there is a permutation $\pi : [n] \to [n]$ with x_i a prefix of y_i for each $i < n$, and for some $i < n$ x_i is a strict prefix of y_i.

It is not hard to see that $\subseteq$ is a well-founded preorder, by reduction to the fact that the prefix relation is a well-founded partial order. As a result, we may duly devise a version of safe (nested) recursion on $\subset$:

Definition 40 (Safe (nested) recursion on permutations of prefixes)**.** We write $\mathsf{NB}^{\subset}(\vec{a})$ for the class of functions over $\vec{a}$ generated from B^- under Safe Composition, and the scheme $\mathsf{snrec}_{\subset}$,

- from $h(a)(\vec{x}; \vec{y})$ over $a, \vec{a}$ define $f(\vec{x}; \vec{y})$ over $\vec{a}$ by:
$$f(\vec{x}; \vec{y}) = h(\lambda \vec{u} \subset \vec{x}, \lambda \vec{v}. f(\vec{u}; \vec{v}))(\vec{x}; \vec{y})$$

as well as the following generalisation of Safe Composition along a Normal Parameter:

(2)′ from $g(\vec{x};\,)$ over $\varnothing$ and $h(\vec{x}, x; \vec{y})$ over $\vec{a}$ define $f(\vec{x}; \vec{y})$ over $\vec{a}$ by $f(\vec{x}; \vec{y}) = h(\vec{x}, g(\vec{x};\,); \vec{y})$.

Recalling Definition 38, we write $\mathsf{SB}^{\subset}(\vec{a})$ to be the restriction of $\mathsf{NB}^{\subset}(\vec{a})$ using only unnested Safe Composition along Safe Parameters. Finally we define $\mathsf{B}^{\subset}(\vec{a})$ to be the restriction of $\mathsf{SB}^{\subset}(\vec{a})$ where every instance of $\mathsf{snrec}_{\subset}$ has the form:

- from $h(a)(\vec{x}; \vec{y})$ over $a, \vec{a}$, define $f(\vec{x}; \vec{y})$ over $\vec{a}$:
$$f(\vec{x}; \vec{y}) = h(\lambda \vec{u} \subset \vec{x}, \lambda \vec{v} \subseteq \vec{y}. f(\vec{u}; \vec{v}))(\vec{x}; \vec{y})$$

We call this latter recursion scheme $\mathsf{srec}_{\subset}$, e.g. if we need to distinguish it from $\mathsf{snrec}_{\subset}$.

Note that the version of safe composition along a normal parameter above differs from the previous one, Item 2 from Definition 36, since the function h is allowed to use oracles. This difference is inessential in terms of computational complexity, as we shall see, but, again, the greater expressivity of $\mathsf{B}^{\subset}$ and $\mathsf{NB}^{\subset}$ will facilitate our overall strategy for characterising CB and CNB, respectively.

Let us take a moment to point out that $\mathsf{NB}^{\subset}(\vec{a}) \supseteq \mathsf{SB}^{\subset}(\vec{a}) \supseteq \mathsf{B}^{\subset}(\vec{a})$ indeed contain only well-defined total functions over the oracles $\vec{a}$, by reduction to induction on the well-founded relation $\subset$.

4.3. **Simultaneous recursion schemes.** Finally, let us establish a standard type of result, that the algebras $\mathsf{B}^{\subset}$, $\mathsf{SB}^{\subset}$ and $\mathsf{NB}^{\subset}$ are closed under simultaneous versions of their recursion schemes.

Definition 41 (Simultaneous schemes)**.** We define schemes $\mathsf{ssrec}_{\subset}$ and $\mathsf{ssnrec}_{\subset}$, respectively, as follows, for arbitrary $\vec{a} = a_1, \ldots, a_k$:

- from $h_i(\vec{a})(\vec{x}; \vec{y})$ over $\vec{a}, \vec{b}$, for $1 \leq i \leq k$, define $f_i(\vec{x}; \vec{y})$ over $\vec{b}$, for $1 \leq i \leq k$, by:
$$f_i(\vec{x}; \vec{y}) = h_i((\lambda \vec{u} \subset \vec{x}, \lambda \vec{v} \subseteq \vec{y}. f_j(\vec{u}; \vec{v}))_{1 \leq j \leq k})(\vec{x}; \vec{y})$$

- from $h_i(\vec{a})(\vec{x};\vec{y})$ over $\vec{a},\vec{b}$, for $1 \leq i \leq k$, define $f_i(\vec{x};\vec{y})$ over $\vec{b}$, for $1 \leq i \leq k$, by:

$$f_i(\vec{x};\vec{y}) = h_i((\lambda\vec{u} \subset \vec{x}, \lambda\vec{v}.f_j(\vec{u};\vec{v}))_{1\leq j\leq k})(\vec{x};\vec{y})$$

Proposition 42. *We have the following:*

(1) If $\vec{f}(\vec{x};\vec{y})$ over $\vec{b}$ are obtained from $\vec{h}(\vec{a})(\vec{x};\vec{y}) \in \mathsf{B}^{\subset}(\vec{a},\vec{b})$ by $\mathsf{ssrec}_{\subset}$, then also $\vec{f}(\vec{x};\vec{y}) \in \mathsf{B}^{\subset}(\vec{b})$.

(2) If $\vec{f}(\vec{x};\vec{y})$ over $\vec{b}$ are obtained from $\vec{h}(\vec{a})(\vec{x};\vec{y}) \in \mathsf{NB}^{\subset}(\vec{a},\vec{b})$ (or $\in \mathsf{SB}^{\subset}(\vec{a},\vec{b})$) by $\mathsf{ssnrec}_{\subset}$, then also $\vec{f}(\vec{x};\vec{y}) \in \mathsf{NB}^{\subset}(\vec{b})$ (resp., $\in \mathsf{SB}^{\subset}(\vec{b})$).

Proof. We only prove Item 1, i.e. that $\mathsf{B}^{\subset}$ is closed under $\mathsf{ssrec}_{\subset}$, but the same argument works for Item 2: just ignore all guards on safe inputs in recursive calls, and our construction preserves unnestedness of Safe Composition along Safe Parameters.

Let $f_i(\vec{x};\vec{y})$ and $h_i(a_1,\ldots,a_k)(\vec{x};\vec{y})$ be as given in Definition 41, and temporarily write $f_j^{\vec{x};\vec{y}}$ for $\lambda\vec{u} \subset \vec{x}, \lambda\vec{v} \subseteq \vec{y}.f_j(\vec{u};\vec{v})$, so we have:

$$f_i(\vec{x};\vec{y}) = h_i(f_1^{\vec{x};\vec{y}},\ldots,f_k^{\vec{x};\vec{y}})(\vec{x};\vec{y})$$

For $i \in \mathbb{N}$, let us temporarily write $\underline{i}$ for i in binary notation[4], and $\vec{i}$ for the list $\underline{i},\underline{i+1},\ldots,\underline{k},\underline{1},\underline{2},\ldots,\underline{i-1}$. Note that, for all $i = 1,\ldots,k$, $\vec{i}$ is a permutation (in fact a rotation) of $\underline{1},\ldots,\underline{k}$.

Now, let $f(\vec{x};\vec{y},\vec{z})$ over oracles $\vec{b}$ be given as follows:

$$f(\vec{x};\vec{y},\vec{z}) := \begin{cases} h_1(f_1^{\vec{x};\vec{y}},\ldots,f_k^{\vec{x};\vec{y}})(\vec{x};\vec{y}) & \vec{z} = \vec{1} \\ \vdots & \\ h_k(f_1^{\vec{x};\vec{y}},\ldots,f_k^{\vec{x};\vec{y}})(\vec{x};\vec{y}) & \vec{z} = \vec{k} \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

Note that this really is a finite case distinction since each of the boundedly many $\vec{i}$ has bounded size, both bounds depending only on k, and so is computable in B^{-} over $\vec{h}$.

By definition, then, we have that $f(\vec{x};\vec{y},\vec{i}) = f_i(\vec{x};\vec{y})$. Moreover note that, for each $j = 1,\ldots,k$, we have,

$$\begin{aligned} f_j^{\vec{x};\vec{y}}(\vec{u}';\vec{v}') &= (\lambda\vec{u} \subset \vec{x}, \lambda\vec{v} \subseteq \vec{y}.f_j(\vec{u};\vec{v}))(\vec{u}';\vec{v}') \\ &= (\lambda\vec{u} \subset \vec{x}, \lambda\vec{v} \subseteq \vec{y}.f(\vec{u};\vec{v},\vec{j}))(\vec{u}';\vec{v}') \\ &= (\lambda\vec{u} \subset \vec{x}, \lambda\vec{v} \subseteq \vec{y}, \lambda\vec{w} \subseteq \vec{z}.f(\vec{u};\vec{v},\vec{w}))(\vec{u}';\vec{v}',\vec{j}) \end{aligned}$$

as long as $\vec{z}$ is some $\vec{i}$, so indeed (6) has the form,

$$f(\vec{x};\vec{y},\vec{z}) = h(\lambda\vec{u} \subset \vec{x}, \lambda\vec{v} \subseteq \vec{y}, \lambda\vec{w} \subseteq \vec{z}.f(\vec{u};\vec{v},\vec{w}))(\vec{x};\vec{y})$$

and $f(\vec{x};\vec{y},\vec{z}) \in \mathsf{B}^{\subset}(\vec{b})$ by $\mathsf{srec}_{\subset}$. Finally, since $f_i(\vec{x};\vec{y}) = f(\vec{x};\vec{y},\vec{i})$, we indeed have that each $f_i(\vec{x};\vec{y}) \in \mathsf{B}^{\subset}(\vec{b})$. □

[4]In fact, any notation will do, but we pick one for concreteness.

5. Characterizations results for function algebras

In this section we characterise the complexities of the function algebras we introduced in the previous section. On the one hand, despite apparently extending B, $\mathsf{B}^{\subset}$ still captures the polynomial-time computable functions. On the other hand, both NB and $\mathsf{NB}^{\subset}$ are shown to capture the elementary functions. All such results rely on a 'bounding lemma' inspired by [BC92].

5.1. **Bounding Lemma.** Bellantoni and Cook showed in [BC92] that any function $f(\vec{x};\vec{y}) \in \mathsf{B}$ satisfies the 'poly-max bounding lemma': there is a polynomial p_f such that:[5]

$$|f(\vec{x};\vec{y})| \leq p_f(|\vec{x}|) + \max|\vec{y}| \tag{7}$$

This provided a suitable invariant for eventually proving that all B-functions were polynomial-time computable.

In this work, inspired by that result, we extend that bounding result to a form suitable for the algebras from the previous section. To this end we establish in the next result a sort of 'elementary-max' bounding lemma that accounts for the usual poly-max bounding as a special case, by appealing to the notion of (un)nested safe composition. What is more, both the statement and the proof are quite subtle due to our algebras' formulation using oracles; we must assume an appropriate bound for the oracles themselves, and the various (mutual) dependencies in the statement are quite delicate.

For us to state and prove the Bounding Lemma, it will be useful to employ the notation $\|\vec{x}\| := \sum |\vec{x}|$.

Lemma 43. *Suppose $f(\vec{x};\vec{y}) \in \mathsf{NB}^{\subset}(\vec{a})$, with $\vec{a} = a_1, \ldots, a_k$. There is an elementary function e_f and a constant $d_f \geq 1$ such that if for $1 \leq i \leq k$,*

$$|a_i(\vec{x}_i;\vec{y}_i)| \leq c_i + d_f \sum_{j \neq i} c_j + \max|\vec{y}_i| \tag{8}$$

for some constants $\vec{c} = c_1, \ldots, c_k$, then we have:

$$|f(\vec{x};\vec{y})| \leq e_f(\|\vec{x}\|) + d_f \sum \vec{c} + \max|\vec{y}| \tag{9}$$

Moreover, if in fact $f(\vec{x};\vec{y}) \in \mathsf{SB}^{\subset}(\vec{a})$, then $d_f = 1$ and $e_f(n)$ is a polynomial.

Unwinding the statement above, note that e_f and d_f depend *only* on the function f itself, not on the constants $\vec{c}$ given for the (mutual) oracle bounds in Equation (8). This is crucial and we shall exploit this in the proof that follows, namely in the case when f is defined by recursion, substituting different values for $\vec{c}$ during an inductive argument. This independency of e_f and d_f from $\vec{c}$ will be made explicit in their definitions in the proof that follows.

While the role of the elementary bounding function e_f is a natural counterpart of p_f in Bellantoni and Cook's bounding lemma, cf. Equation (7), the role of d_f is perhaps slightly less clear. Morally, d_f represents the amount of 'nesting' in the definition of f, increasing whenever oracle calls are substituted into arguments for other oracles. Hence, if f uses only unnested Safe Composition, then $d_f = 1$ as required. In the argument that follows, it will only be important to distinguish whether $d_f = 1$ or not, since d_f will form the base of an exponent when defining e_f in the case when f is defined by safe recursion.

[5] Recall that, for $\vec{x} = x_1, \ldots, x_n$, we write $|\vec{x}|$ for $|x_1|, \ldots, |x_n|$.

Let us also point out that we may relativise the statement of the lemma to any set of oracles including those which $f(\vec{x};\vec{y})$ is over. In particular, if $f(\vec{x};\vec{y})$ is over no oracles, then we may realise Equation (8) vacuously by choosing $\vec{a} = \varnothing$ and we would obtain that $|f(\vec{x};\vec{y})| \leq e_f(\|\vec{x}\|) + \max|\vec{y}|$. More interestingly, in the case when $f(\vec{x};\vec{y})$ is just, say, $a_i(\vec{x};\vec{y})$, we may choose to set $\vec{a} = a_i$ or $\vec{a} = a_1, \ldots, a_n$ in the above lemma, yielding different bounds in each case. We shall exploit this in inductive hypotheses in the proof that follows (typically when we write 'WLoG').

Proof of Lemma 43. We will proceed by induction on the definition of $f(\vec{x};\vec{y})$, always assuming that we have $\vec{c}$ satisfying Equation (8).

Throughout the argument we shall actually construct $e_f(n)$ that are *monotone* elementary functions (without loss of generality) and exploit this invariant. In fact $e_f(n)$ will always be generated by composition from $0, \mathsf{s}, +, \times, x^y$ and projections. If $f(\vec{x};\vec{y}) \in \mathsf{SB}^{\subset}(\vec{a})$ then $e_f(n)$ will be in the same algebra without exponentiation, x^y, i.e. it will be a polynomial with only non-negative coefficients. This property will be made clear by the given explicit definitions of $e_f(n)$ throughout the argument.

If $f(\vec{x};\vec{y})$ is an initial function then it suffices to set $e_f(n) := 1 + n$ and $d_f := 1$.

If $f(\vec{x};\vec{y}) = a_i(\vec{x};\vec{y})$ then it suffices to set $e_f(n) := 0$ and $d_f := 1$.

If $f(\vec{x};\vec{y}) = h(\vec{x};\vec{y}, g(\vec{x};\vec{y}))$, let e_h, e_g, d_h, d_g be obtained from the inductive hypothesis. We have,

$$\begin{array}{rcl} |f(\vec{x};\vec{y})| & = & |h(\vec{x};\vec{y}, g(\vec{x};\vec{y}))| \\ & \leq & e_h(\|\vec{x}\|) + d_h \sum \vec{c} + \max(|\vec{y}|, |g(\vec{x};\vec{y})|) \\ & \leq & e_h(\|\vec{x}\|) + d_h \sum \vec{c} + e_g(\|\vec{x}\|) + d_g \sum \vec{c} + \max|\vec{y}| \end{array}$$

so we may set $e_f(n) := e_h(n) + e_g(n)$ and $d_f := d_h + d_g$.

For the 'moreover' clause, note that if $f(\vec{x};\vec{y})$ uses only unnested Safe Composition, then one of g or h does not have oracles, and so we can assume WLoG that either the term $d_h \sum \vec{c}$ or the term $d_g \sum \vec{c}$ above does not occur above, and we set $d_f := d_g$ or $d_f := d_h$, respectively. In either case we obtain $d_f = 1$ by the inductive hypothesis, as required.

If $f(\vec{x};\vec{y}) = h(\vec{x}, g(\vec{x};);\vec{y})$, let e_h, e_g, d_h, d_g be obtained from the inductive hypothesis. Note that, by definition of Safe Composition along a normal parameter, we must have that g has no oracles, and so in fact $|g(\vec{x};)| \leq e_g(\|\vec{x}\|)$. We thus have,

$$\begin{array}{rcl} |f(\vec{x};\vec{y})| & = & |h(\vec{x}, g(\vec{x};);\vec{y})| \\ & \leq & e_h(\|\vec{x}\|, |g(\vec{x};)|) + d_h \sum \vec{c} + \max|\vec{y}| \\ & \leq & e_h(\|\vec{x}\|, e_g(\|\vec{x}\|)) + d_h \sum \vec{c} + \max|\vec{y}| \end{array}$$

so we may set $e_f(n) := e_h(n, e_g(n))$ and $d_f := d_h$. For the 'moreover' clause, note that by the inductive hypothesis e_h, e_g are polynomials and $d_h = 1$, so indeed e_f is a polynomial and $d_f = 1$.

Finally, if $f(\vec{x};\vec{y}) = h(\lambda\vec{u} \subset \vec{x}, \lambda\vec{v}.f(\vec{u};\vec{v}))(\vec{x};\vec{y})$, let e_h, d_h be obtained from the inductive hypothesis. We claim that it suffices to set $d_f := d_h$ and $e_f(n) := nd_h^n e_h(n)$. Note that, for the 'moreover' clause, if $d_h = 1$ then also $d_h^n = 1$ and so indeed $e_f(n)$ is a polynomial if $d_h = 1$ and $e_h(n)$ is a polynomial.

First, let us calculate the following invariant, for $n > 0$:

$$
\begin{array}{rcll}
e_f(n) & = & nd_h^n e_h(n) & \\
& = & d_h^n e_h(n) + (n-1)d_h^n e_h(n) & \\
& = & d_h^n e_h(n) + d_h(n-1)d_h^{n-1} e_h(n) & \\
& \geq & e_h(n) + d_h(n-1)d_h^{n-1} e_h(n) & \text{since } d_h \geq 1 \\
& \geq & e_h(n) + d_h(n-1)d_h^{n-1} e_h(n-1) & \text{since } e_h \text{ is monotone} \\
& \geq & e_h(n) + d_h e_f(n-1) & \text{by definition of } e_f
\end{array} \tag{10}
$$

Now, to show that Equation (9) bounds for the f, e_f, d_f at hand, we proceed by a sub-induction on $\|\vec{x}\|$. For the base case, when $\|\vec{x}\| = 0$ (and so, indeed, $\vec{x} = \vec{0}$), note simply that $\lambda\vec{u} \subset \vec{x}, \lambda\vec{v}.f(\vec{u};\vec{v})$ is the constant function 0, and so we may appeal to the main inductive hypothesis for $h(a)$ setting the corresponding constant c for a to be 0 to obtain,

$$
\begin{array}{rcl}
|f(\vec{0};\vec{y})| & = & |h(0)(\vec{0};\vec{y})| \\
& \leq & e_h(0) + d_h \sum \vec{c} + \max|\vec{y}| \\
& \leq & e_f(0) + d_f \sum \vec{c} + \max|\vec{y}|
\end{array}
$$

as required. For the sub-inductive step, let $\|\vec{x}\| > 0$. Note that, whenever $\vec{u} \subset \vec{x}$ we have $\|\vec{u}\| < \|\vec{x}\|$ and so, by the sub-inductive hypothesis and monotonicity of e_f we have:

$$|f(\vec{u};\vec{v})| \leq e_f(\|\vec{x}\| - 1) + d_f \sum \vec{c} + \max|\vec{v}|$$

Now we may again appeal to the main inductive hypothesis for $h(a)$ by setting $c = e_f(\|\vec{x}\| - 1)$ to be the corresponding constant for $a = \lambda\vec{u} \subset \vec{x}, \lambda\vec{v}.f(\vec{u};\vec{v})$. We thus obtain:

$$
\begin{array}{rcll}
|f(\vec{x};\vec{y})| & = & |h(\lambda\vec{u} \subset \vec{x}, \lambda\vec{v}.f(\vec{u};\vec{v}))(\vec{x};\vec{y})| & \\
& \leq & e_h(\|\vec{x}\|) + d_h c + d_h \sum \vec{c} + \max|\vec{y}| & \text{by main IH} \\
& \leq & (e_h(\|\vec{x}\|) + d_h e_f(\|\vec{x}\| - 1)) + d_h \sum \vec{c} + \max|\vec{y}| & \text{since } c = e_f(\|\vec{x}\| - 1) \\
& \leq & e_f(\|\vec{x}\|) + d_h \sum \vec{c} + \max|\vec{y}| & \text{by Equation (10)} \\
& \leq & e_f(\|\vec{x}\|) + d_f \sum \vec{c} + \max|\vec{y}| & \text{since } d_f = d_h
\end{array}
$$

This completes the proof of the Bounding Lemma. □

Before moving on to our soundness results, let us first present an important consequence of the bounding lemma that will be important for characterising $\mathsf{NB}^{\subset}$.

Proposition 44. *Let $f(\vec{a})(\vec{x};\vec{y}) \in \mathsf{NB}^{\subset}(\vec{a})$ and let e_f and d_f be as constructed in the proof of Lemma 43 and write:*

$$m_f^{\vec{c}}(\vec{x},\vec{y}) \;:=\; e_f(\|\vec{x}\|) + d_f \sum \vec{c} + \max|\vec{y}|$$

If there are constants $\vec{c}$ such that $\vec{a}$ satisfy Equation (8)*, then:*[6]

$$f(\vec{a})(\vec{x};\vec{y}) \;=\; f(\lambda\vec{x}_i.\lambda|\vec{y}_i| \leq m_f^{\vec{c}}(\vec{x},\vec{y}).a_i(\vec{x}_i;\vec{y}_i))_i(\vec{x};\vec{y}) \tag{11}$$

Note that the point of the above proposition is somewhat dual to that of the bounding lemma: while the latter bounds the *output* of a function, the former bounds the *inputs*. Nonetheless, the argument follows essentially the same structure as that of the Bounding Lemma 43.

[6]To be clear, here we write $|\vec{y}_i| \leq m_f^{\vec{c}}(\vec{x},\vec{y})$ here as an abbreviation for $\{|y_{ij} \leq m_f^{\vec{c}}(\vec{x},\vec{y})\}_j$.

Proof. We proceed by induction on the definition of $f(\vec{a})(\vec{x};\vec{y})$, always making explicit the oracles of a function.

The initial functions and oracle calls are immediate, due to the 'max $|\vec{y}|$' term in Equation (11).

If $f(\vec{a})(\vec{x};\vec{y}) = h(\vec{a})(\vec{x};\vec{y}, g(\vec{a})(\vec{x};\vec{y}))$ then, by the inductive hypothesis for $h(\vec{a})$, any oracle call from $h(\vec{a})$ only take safe inputs of lengths:

$$\begin{array}{ll} \leq & e_h(\|\vec{x}\|) + d_h \sum \vec{c} + \max(|\vec{y}|, |g(\vec{a})(\vec{x};\vec{y})|) \\ \leq & e_h(\|\vec{x}\|) + d_h \sum \vec{c} + e_g(\|\vec{x}\|) + d_g \sum \vec{c} + \max|\vec{y}| \quad \text{by Bounding Lemma 43} \\ \leq & (e_h(\|\vec{x}\| + e_g(\|\vec{x}\|)) + (d_h + d_g) \sum \vec{c} + \max|\vec{y}| \\ \leq & e_f(\|\vec{x}\|) + d_f \sum \vec{c} + \max|\vec{y}| \end{array}$$

Note that any oracle call from $g(\vec{a})$ will still only take safe inputs of lengths $\leq e_g(\|\vec{x}\|) + d_g \sum \vec{c} + \max|\vec{y}|$, by the inductive hypothesis, and e_g and d_g are bounded above by e_f and d_f respectively.

If $f(\vec{a})(\vec{x};\vec{y}) = h(\vec{a})(\vec{x}, g(\varnothing)(\vec{x};);\vec{y})$ then, by the inductive hypothesis, any oracle call will only take safe inputs of lengths:

$$\begin{array}{ll} \leq & e_h(\|\vec{x}\| + |g(\varnothing)(\vec{x};)|) + d_h \sum \vec{c} + \max|\vec{y}| \\ \leq & e_h(\|\vec{x}\| + e_g(\|\vec{x}\|)) + d_h \sum \vec{c} + \max|\vec{y}| \qquad \text{by Bounding Lemma 43} \\ \leq & e_f(\|\vec{x}\|) + d_f \sum \vec{c} + \max|\vec{y}| \end{array}$$

Finally suppose $f(\vec{a})(\vec{x};\vec{y}) = h(\vec{a}, \lambda\vec{u} \subset \vec{x}, \lambda\vec{v}.f(\vec{a})(\vec{u};\vec{v}))(\vec{x};\vec{y})$. We proceed by a sub-induction on $\|\vec{x}\|$. Note that, since $\vec{u} \subset \vec{x} \implies \|\vec{u}\| < \|\vec{x}\|$, we immediately inherit from the inductive hypothesis the appropriate bound on safe inputs for oracle calls from $\lambda\vec{u} \subset \vec{x}, \lambda\vec{v}.f(\vec{a})(\vec{u};\vec{v})$.

Now, recall from the Bounding Lemma 43, whenever $\vec{u} \subset \vec{x}$ (and so $\|\vec{u}\| < \|\vec{x}\|$), we have $|f(\vec{u};\vec{v})| \leq e_f(\|\vec{x}\| - 1) + d_f \sum \vec{c} + \max|\vec{v}|$. So by setting $c = e_f(\|\vec{x}\| - 1)$ in the inductive hypothesis for $h(\vec{a}, a)$, with $a = \lambda\vec{u} \subset \vec{x}, \lambda\vec{v}.f(\vec{a})(\vec{u};\vec{v})$, any oracle call from $h(\vec{a}, a)$ will only take safe inputs of lengths:

$$\begin{array}{ll} \leq & e_h(\|\vec{x}\|) + d_h e_f(\|\vec{x}\| - 1) + d_h \sum \vec{c} + \max|\vec{y}| \\ \leq & e_f(\|\vec{x}\|) + d_f \sum \vec{c} + \max|\vec{y}| \qquad \text{by Equation (10)} \end{array}$$

This completes the proof. □

5.2. **Soundness results.** In this subsection we show that the function algebras $\mathsf{B}^{\subset}$ and $\mathsf{NB}^{\subset}$ (as well as NB) capture precisely the classes **FPTIME** and **FELEMENTARY**, respectively. We start with $\mathsf{B}^{\subset}$:

Theorem 45. *Suppose $\vec{a}$ satisfies Equation* (8) *for some constants $\vec{c}$. We have the following:*

*(1) if $f(\vec{x};\vec{y}) \in \mathsf{B}^{\subset}(\vec{a})$ then $f(\vec{x},\vec{y}) \in$ **FPTIME**$(\vec{a})$.*
*(2) if $f(\vec{x};\vec{y}) \in \mathsf{NB}^{\subset}(\vec{a})$ then $f(\vec{x},\vec{y}) \in$ **FELEMENTARY**$(\vec{a})$.*

Before we give the proof note in particular that, for $f(\vec{x};\vec{y})$ in $\mathsf{B}^{\subset}$ or $\mathsf{NB}^{\subset}$, i.e. not using any oracles, we immediately obtain membership in **FPTIME** or **FELEMENTARY**, respectively. However, the reliance on intermediate oracles during a function definition causes some difficulties that we must take into account. At a high level, the idea is to use the Bounding Lemma, namely its consequence Proposition 44, to replace certain oracle calls with explicit appropriately bounded functions computing their graphs.

Proof. We proceed by induction on the definition of $f(\vec{x};\vec{y})$.

Each initial function is polynomial-time computable, and each (relativised) complexity class considered is under composition, so it suffices to only consider the respective recursion schemes. We shall focus first on the case of $\mathsf{B}^{\subset}(\vec{a})$, Item 1 above, so that e_f is a polynomial and $d_f = 1$ (since, in particular, $\mathsf{B}^{\subset}(\vec{a}) \subseteq \mathsf{SB}^{\subset}(\vec{a})$).

Suppose we have $h(a)(\vec{x};\vec{y}) \in \mathsf{B}^{\subset}(a,\vec{a})$ and let:

$$f(\vec{x};\vec{y}) = h(\lambda\vec{u} \subset \vec{x}, \lambda\vec{v} \subseteq \vec{y}.f(\vec{u};\vec{v}))(\vec{x};\vec{y})$$

We start by making some observations:

(1) First, note that $|f(\vec{x};\vec{y})| \leq e_f(|\vec{x}|) + d_f \sum \vec{c} + \max|\vec{y}|$, by the Bounding Lemma 43, and so $|f(\vec{x};\vec{y})|$ is polynomial in $|\vec{x},\vec{y}|$.

(2) Second, note that the set $[\vec{x};\vec{y}] := \{(\vec{u},\vec{v}) \mid \vec{u} \subset \vec{x}, \vec{v} \subseteq \vec{y}\}$ has size polynomial in $|\vec{x},\vec{y}|$:
   - write $\vec{x} = x_1,\ldots,x_m$ and $\vec{y} = y_1,\ldots,y_n$.
   - Each x_i and y_j have only linearly many prefixes, and so there are at most $|x_1|\cdot\cdots\cdot|x_m||y_1|\cdot\cdots\cdot|y_n| \leq \|\vec{x},\vec{y}\|^{m+n}$ many choices of prefixes for all the arguments $\vec{x},\vec{y}$. (This is a polynomial since m and n are global constants).
   - Additionally, there are $m!$ permutations of the arguments $\vec{x}$ and $n!$ permutations of the arguments $\vec{y}$. Again, since m and n are global constants, we indeed have $|[\vec{x};\vec{y}]| = O(\|\vec{x},\vec{y}\|^{m+n})$, which is polynomial in $|\vec{x},\vec{y}|$.

We describe a polynomial-time algorithm for computing $f(\vec{x};\vec{y})$ (over oracles $\vec{a}$) by a sort of 'course-of-values' recursion on the order $\subset \times \subseteq$ on $[\vec{x};\vec{y}]$.

First, for convenience, temporarily extend $\subset \times \subseteq$ to a total well-order on $[\vec{x};\vec{y}]$, and write S for the associated successor function. Note that S can be computed in polynomial-time from $[\vec{x};\vec{y}]$.

Define $F(\vec{x},\vec{y}) := \langle f(\vec{u};\vec{v})\rangle_{\vec{u}\subset\vec{x},\vec{v}\subseteq\vec{y}}$, i.e. it is the graph of $\lambda\vec{u} \subset \vec{x}, \lambda\vec{v} \subseteq \vec{y}.f(\vec{u};\vec{v})$ that we shall use as a 'lookup table'. Note that $|F(\vec{x},\vec{y})|$ is polynomial in $|\vec{x},\vec{y}|$ by Item 1 and Item 2 above. Now, we can write:[7]

$$\begin{aligned} F(S(\vec{x},\vec{y})) &= \langle f(S(\vec{x},\vec{y})), F(\vec{x},\vec{y})\rangle \\ &= \langle h(F(\vec{x},\vec{y}))(\vec{x};\vec{y}), F(\vec{x},\vec{y})\rangle \end{aligned}$$

Again by Item 2 (and since F is polynomially bounded), this recursion terminates in polynomial-time. We may now simply calculate $f(\vec{x};\vec{y})$ as $h(F(\vec{x},\vec{y}))(\vec{x};\vec{y})$.

The argument for $\mathsf{NB}^{\subset}$ is similar, though we need not be as careful about computing the size of the lookup tables (F above) for recursive calls. The key idea is to use the Bounding Lemma, namely its consequence Proposition 44, to bound the safe inputs of recursive calls so that we can adequately store the lookup table for previous values. □

5.3. **Completeness and characterisations.** Note that we have the following as an immediate consequence of the soundness result, Theorem 45 Item 1:

Corollary 46. *The following statements are equivalent:*

(1) $f(\vec{x};) \in \mathsf{B}$
(2) $f(\vec{x};) \in \mathsf{B}^{\subset}$
(3) $f(\vec{x}) \in \mathbf{FPTIME}$

[7]Here, as abuse of notation, we are now simply identifying $F(\vec{x};\vec{y})$ with $\lambda\vec{u} \subset \vec{x}, \lambda\vec{v} \subseteq \vec{y}.f(\vec{x};\vec{y})$.

Proof sketch. (1) $\implies$ (2) is trivial, and (2) $\implies$ (3) is given by Theorem 45.(1). Finally, (3) $\implies$ (1) is from [BC92], stated in Theorem 2 earlier. □

The remainder of this subsection is devoted to establishing a similar equivalence for NB, $\mathsf{NB}^{\subset}$ and **FELEMENTARY**. To this end, we naturally require the following result:

Theorem 47. *If* $f(\vec{x}) \in$ **FELEMENTARY** *then* $f(\vec{x};) \in \mathsf{NB}$.

The proof can be divided into three steps:

- We define an equivalent formulation of the class **FELEMENTARY** for binary notation, having an exponential growth rate function $\varepsilon^1(x)$, whence one may define arbitrary elementary growth rate by setting $\varepsilon^{m+1}(x) := \varepsilon^1(\varepsilon^m(x))$. Now we show that any function $f(\vec{x}) \in$ **FELEMENTARY** in binary notation is bounded above by $\varepsilon^m(\max(\vec{x}))$ for some m.
- We show that for any function $f(\vec{x}) \in$ **FELEMENTARY** in binary notation there are a function $f^*(x;\vec{x}) \in \mathsf{NB}$ and a monotone function[8] $e_f \in$ **FELEMENTARY** such that for all integers $\vec{x}$ and all $w \geq e_f(\vec{x})$ we have $f^*(w;\vec{x}) = f(\vec{x})$.
- Now, given the function $\mathsf{ex}(x;y)$ in (4), we construct the function $\mathsf{ex}^m(x;)$ by induction on m:

$$\mathsf{ex}^1(x;) = \mathsf{ex}(x;1)$$
$$\mathsf{ex}^{m+1}(x;) = \mathsf{ex}^m(\mathsf{ex}^1(x;);)$$

  so that, by increasing m, $\mathsf{ex}^m(x;)$ achieves arbitrary elementary growth rate, just like ε^m. If $f(\vec{x}) \in$ **FELEMENTARY**, by the previous point we obtain a function $f^*(w;\vec{x}) \in \mathsf{NB}$ and a monotone function $e_f \in$ **FELEMENTARY**. By the first point, there exists $m \geq 1$ such that $e_f(\vec{x}) \leq \mathsf{ex}^m(\max_{\sharp\vec{x}}(\vec{x};);)$, where $\max(\vec{x};)$ in NB. Therefore, $f(\vec{x};) = f^*(\mathsf{ex}^m(\max_{\sharp\vec{x}}(\vec{x};););\vec{x}) \in \mathsf{NB}$.

In particular, the first point is folklore, while the proof technique adopted for proving the last two points is well-known since [BC92], and has been adapted to the case of the elementary functions by [WW99].

Now, by the same argument as for Corollary 46, only using Theorem 47 above instead of appealing to [BC92], we can give our main characterisation result for algebras for elementary computation:

Corollary 48. *The following statement are equivalent:*

(1) $f(\vec{x};) \in \mathsf{NB}$,
(2) $f(\vec{x};) \in \mathsf{NB}^{\subset}$,
(3) $f(\vec{x}) \in$ **FELEMENTARY**.

6. Completeness for circular systems

We now return our attention to the circular systems CB and CNB that we introduced in Section 3. We will address the complexity of their definable functions by 'sandwiching' them between function algebras of Section 4, given their characterisations in the previous section.

In particular, in this section we show that CB is 'complete' for **FPTIME** and that CNB is complete for **FELEMENTARY**.

[8]Bear in mind that the function e_f here is not necessarily the same as that of Lemma 43 in the previous section.

6.1. **CB contains all polynomial-time functions.** To show that CB contains all polynomial-time functions, we simulate Bellantoni and Cook's algebra:

Theorem 49. *If* $f(\vec{x};\vec{y}) \in \mathsf{B}$ *then* $f(\vec{x};\vec{y}) \in \mathsf{CB}$.

Proof. By Proposition 6 it suffices to show that for any B-derivation $\mathcal{D}$ there is a CB-coderivation $\mathcal{D}^*$ such that $f_{\mathcal{D}}(\vec{x};\vec{y}) = f_{\mathcal{D}^*}(\vec{x},\vec{y})$. The proof is by induction on $\mathcal{D}$. The only non-trivial case is when $\mathcal{D}$ is the following derivation:

$$\mathsf{srec}_{\Box}\ \frac{\mathcal{D}_0 \quad \mathcal{D}_1 \quad \mathcal{D}_2}{\Gamma \Rightarrow N \qquad \Box N, \Gamma, N \Rightarrow N \qquad \Box N, \Gamma, N \Rightarrow N}{\Box N, \Gamma \Rightarrow N}$$

We define $\mathcal{D}^*$ as follows:

$$\mathsf{cond}_{\Box}\ \frac{\mathcal{D}_0^* \qquad \mathsf{cut}_N\ \dfrac{\mathsf{cond}_{\Box}\ \dfrac{\vdots}{\Box N, \Box\vec{N}, \vec{N} \Rightarrow N}\bullet \qquad \mathcal{D}_i^*}{\Box N, \Box\vec{N}, \vec{N}, N \Rightarrow N}\ i=1,2}{\Box\vec{N}, \vec{N} \Rightarrow N \qquad \Box N, \Box\vec{N}, \vec{N} \Rightarrow N}\bullet$$

where we identify the coderivations corresponding to the second and the third premise of the conditional rule, as they only differ on the sub-coderivation $\mathcal{D}_i^*$ ($i = 1$ for the former and $i = 2$ for the latter). The above coderivation is clearly safe and left-leaning, by the inductive hypotheses for $\mathcal{D}_0, \mathcal{D}_1, \mathcal{D}_2$. To see that it is progressing, note that any infinite branch is either eventually entirely in $\mathcal{D}_0^*$, $\mathcal{D}_1^*$ or $\mathcal{D}_2^*$, in which case it is progressing by the inductive hypotheses, or it simply loops on $\bullet$ forever, in which case there is a progressing thread along the blue $\Box N$.

Moreover, the equational program associated with $\mathcal{D}$ can be is equivalent to:

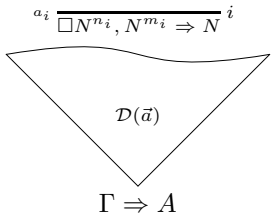
$$\begin{array}{rcll} f_{\mathcal{D}_\epsilon^*}(0,\vec{x};\vec{y}) & := & f_{\mathcal{D}_0^*}(\vec{x};\vec{y}) & \\ f_{\mathcal{D}_\epsilon^*}(\mathsf{s}_0 x,\vec{x};\vec{y}) & := & f_{\mathcal{D}_1^*}(x,\vec{x};\vec{y}, f_{\mathcal{D}_\epsilon^*}(x,\vec{x};\vec{y})) & \text{if } x \neq 0 \\ f_{\mathcal{D}_\epsilon^*}(\mathsf{s}_1 x,\vec{x};\vec{y}) & := & f_{\mathcal{D}_2^*}(x,\vec{x};\vec{y}, f_{\mathcal{D}_\epsilon}(x,\vec{x};\vec{y})) & \end{array}$$

so that $f_{\mathcal{D}}(\vec{x};\vec{y}) = f_{\mathcal{D}^*}(\vec{x},\vec{y})$. □

6.2. **CNB contains all elementary functions.** Similarly to the case for CB, to show that CNB is complete for elementary functions, we shall simulate our nested algebra NB. First, we need to introduce the notion of *oracle* for B^--coderivations.

Definition 50 (Oracles for coderivations)**.** Let $\vec{a} = a_1, \ldots, a_n$ be a set of safe-normal functions. A $\mathsf{B}^-(\vec{a})$-coderivations is just a usual B^--coderivation that may use initial sequents of the form $a_i\ \dfrac{}{\Box N^{n_i}, N^{m_i} \Rightarrow N}$, when a_i takes n_i normal and

m_i safe inputs. We write:

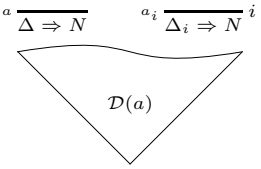


for a coderivation $\mathcal{D}$ with initial sequents among $a_i \, \overline{\Box N^{n_i}, N^{m_i} \Rightarrow N}$, with $i = 1, \ldots, n$. We write $\mathsf{CNB}(\vec{a})$ for the set of CNB-coderivations with initial functions $\vec{a}$. We may sometimes omit indicating some oracles $\vec{a}$ if it is clear from context.

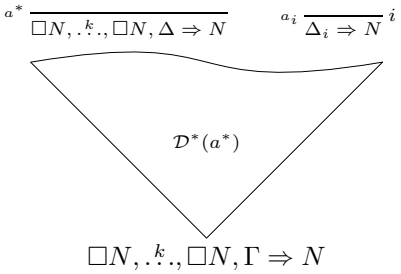
The semantics of such coderivations and the notion of $\mathsf{CNB}(\vec{a})$-definability are as expected, with coderivations representing functions over the oracles $\vec{a}$, and $f_{\mathcal{D}(\vec{a})} \in \mathsf{CNB}(\vec{a})$ denoting the induced interpretation of $\mathcal{D}(\vec{a})$.

Before giving our main completeness result for CNB, we need the following lemma allowing us to 'pass' parameters to oracle calls. It is similar to the notion $\mathcal{D}^{\vec{\rho}}$ from [Das21, Lemma 42], only we must give a more refined argument due to the unavailability of contraction in our system.

Lemma 51. *Let $\mathcal{D}(a)$ be a regular coderivation over initial sequents $\vec{a}, a$ with shape:*



where Γ and Δ are lists of non-modal formulas, and the path from the conclusion to each initial sequent a does not contain $\mathsf{cut}_\Box$-steps, $\Box_l$-steps and the leftmost premise of a $\mathsf{cond}_\Box$-step. Then, there exists an a^* and a regular coderivation $\mathcal{D}^*(a^*)$ with shape:



such that:

- $f_{\mathcal{D}^*(a^*)}(\vec{x}; \vec{y}) = f_{\mathcal{D}(a(\vec{x}))}(\vec{x}; \vec{y})$*;*
- *there exists a $\Box N$-thread from the j^{th} modal formula in the context of the conclusion to the j^{th} modal formula in the context of any occurrence of the initial sequent a^* in $\mathcal{D}^*(a^*)$, for $1 \leq j \leq k$.*

Moreover, if $\mathcal{D}(a)$ is progressing, safe or left-leaning, then $\mathcal{D}^*(a^*)$ is also progressing, safe or left-leaning, respectively.

Proof sketch. Let us consider a path B from the root of $\mathcal{D}(a)$ to an occurrence of the initial sequent a. Since the conclusion of $\mathcal{D}(a)$ contains k modal formulas, and B cannot cross $\mathsf{cut}_\Box$-steps, B contains exactly k $\Box N$-threads, and all such threads start from the root. Moreover, since a has only non-modal formulas and B cannot cross $\Box_l$-steps or the leftmost premise of a $\mathsf{cond}_\Box$-step, we conclude that each of the k $\Box N$-threads must end in the principal formula of a $\mathsf{w}_\Box$-step. For each j we remove the corresponding $\mathsf{w}_\Box$-step in B we add an extra $\Box N$ to the antecedent of all higher sequents in B (this operation may require us to introduce weakening steps for other branches of the proof). By repeatedly applying the above procedure for each possible path from the root of $\mathcal{D}(a)$ to an initial sequent a we obtain a coderivation with the desired properties. □

Finally we can give our main simulation result for CNB:

Theorem 52. *If $f(\vec{x};\vec{y}) \in \mathsf{NB}$ then $f(\vec{x};\vec{y}) \in \mathsf{CNB}$.*

Proof. We show by induction on $f(\vec{x};\vec{y}) \in \mathsf{NB}(\vec{a})$ that there is a $\mathsf{CNB}(\vec{a})$-coderivation $\mathcal{D}_f$ such that:

(1) $f_{\mathcal{D}_f}(\vec{x};\vec{y}) = f(\vec{x};\vec{y})$;
(2) the path from the conclusion of $\mathcal{D}_f$ to each initial sequent a_i does not contain $\mathsf{cut}_\Box$-steps, $\Box_l$-steps and the leftmost premise of a $\mathsf{cond}_\Box$-step.

When $f(\vec{x};\vec{y})$ is an initial function the definition of $\mathcal{D}^*$ is straightforward, as $\vec{a} = \varnothing$.

If $f(\vec{x};\vec{y}) = a_i(\vec{x};\vec{y})$ then $\mathcal{D}_f$ is the initial sequent $\dfrac{}{\Box\vec{N}, \vec{N} \Rightarrow N}\, a_i$.

Suppose that $f(\vec{x};\vec{y}) = h(\vec{x}, g(\vec{x};);\vec{y})$ with $g(\vec{x};) \in \mathsf{NB}(\varnothing)$ and $h(\vec{x}, z;\vec{y}) \in \mathsf{NB}(\varnothing)$. Then f can be $\mathsf{NB}(\varnothing)$-defined by:

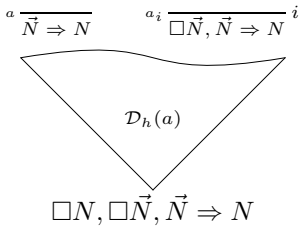
$$\dfrac{\dfrac{\varnothing\ \mathcal{D}_g}{\Box\vec{N} \Rightarrow N}}{\Box\vec{N} \Rightarrow \Box N}\,\Box_r \qquad \dfrac{\varnothing\ \mathcal{D}_h}{\Box N, \Box\vec{N}, \vec{N} \Rightarrow N}}{\Box\vec{N}, \vec{N} \Rightarrow N}\,\mathsf{cut}_\Box$$

Suppose $f(\vec{x};\vec{y}) = h(\vec{x};\vec{y}, g(\vec{x};\vec{y}))$. Then f is $\mathsf{NB}(\vec{a})$-defined by:

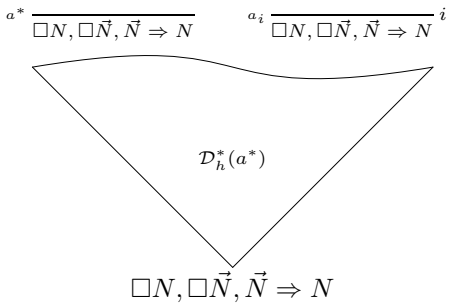
$$\dfrac{\dfrac{a_i \dfrac{}{\Box\vec{N}, \vec{N} \Rightarrow N}\, i\ \ \mathcal{D}_g}{\Box\vec{N}, \vec{N} \Rightarrow N} \qquad \dfrac{a_i \dfrac{}{\Box\vec{N}, \vec{N} \Rightarrow N}\, i\ \ \mathcal{D}_h}{\Box\vec{N}, \vec{N}, N \Rightarrow N}}{\Box\vec{N}, \vec{N} \Rightarrow N}\,\mathsf{cut}_N$$

Note that, while we introduce a $\mathsf{cut}_\Box$ here, there crucially remains no $\mathsf{cut}_\Box$ between the conclusion and an oracle sequent, thanks to the condition that g and h are over no oracles.

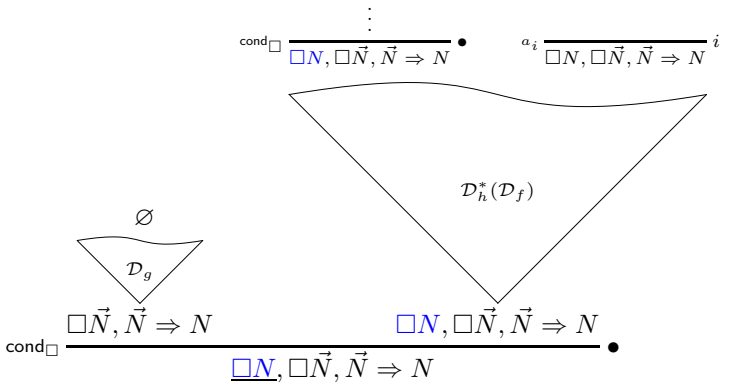
Finally, suppose that $f(x,\vec{x};\vec{y})$ is obtained from $g(\vec{x};\vec{y})$ over $\varnothing$, and $h(a)(x,\vec{x};\vec{y})$ over $a,\vec{a}$ by snrec. Then, $f(0,\vec{x};\vec{y}) = g(\vec{x};\vec{y})$ and $f(\mathsf{s}_i x,\vec{x};\vec{y}) = h(\lambda\vec{v}.f(x,\vec{x};\vec{v}))(x,\vec{x};\vec{y})$. Note that a has same type as $\lambda\vec{v}.f(x,\vec{x};\vec{v})$, so it is a function taking safe arguments only. Thus by induction hypothesis, $h(a)(x,\vec{x};\vec{y})$ is $\mathsf{NB}(\vec{a},a)$-defined by:



where the path from the conclusion of $\mathcal{D}_h(a)$ to each initial sequent a_i does not contain $\mathsf{cut}_\Box$-steps, $\Box_l$-steps and the leftmost premise of a $\mathsf{cond}_\Box$-step. By Lemma 51 we obtain the following coderivation:



where $f_{\mathcal{D}^*_h(a^*)}(x,\vec{x};\vec{y}) = f_{\mathcal{D}_h(a(x,\vec{x}))}(x,\vec{x};\vec{y})$ and there exists a $\Box N$-thread from the j-th modal formula in the antecedent of the conclusion to the j-th modal formula in the antecedent of any a^* initial sequent in $\mathcal{D}^*(a^*)$. We define $\mathcal{D}_f$ as follows:



where we identify the sub-coderivations corresponding to the second and the third premises of the conditional rule. By construction and induction hypothesis, the above coderivation is regular and safe. To see that it is progressing, note that any infinite branch B either hits $\bullet$ infinitely often, in which case there is a progressing thread along the blue $\Box N$, by the properties of $\mathcal{D}^*_h$ inherited from Lemma 51, or B

shares a tail with an infinite branch of $\mathcal{D}_g$ or $\mathcal{D}^*_h(a^*)$, which are progressing by the inductive hypotheses.

We show that $\mathcal{D}_f$ $\mathsf{NB}(\vec{a})$-defines f by induction on x. For the base case, $f_{\mathcal{D}^*_f}(0,\vec{x};\vec{y}) = f_{\mathcal{D}_g}(\vec{x};\vec{y}) = g(\vec{x};\vec{y}) = f(0,\vec{x};\vec{y})$. For the inductive step:

$$\begin{aligned} f_{\mathcal{D}^*_f}(\mathsf{s}_i x,\vec{x};\vec{y}) &= f_{\mathcal{D}^*_h(\lambda\vec{v}.f_{\mathcal{D}^*_f}(x,\vec{x};\vec{v}))}(x,\vec{x};\vec{y}) \\ &= f_{\mathcal{D}^*_h(\lambda\vec{v}.f(x,\vec{x};\vec{v}))}(x,\vec{x};\vec{y}) \\ &= f_{\mathcal{D}_h(\lambda\vec{v}.f(x,\vec{x};\vec{v}))}(x,\vec{x};\vec{y}) \\ &= h(\lambda\vec{v}.f(x,\vec{x};\vec{v}))(x,\vec{x};\vec{y}) \\ &= f(\mathsf{s}_i x,\vec{x};\vec{y}). \end{aligned}$$

This completes the proof. □

7. Soundness for circular systems

In this section we define a translation of CNB-coderivations into functions of $\mathsf{NB}^{\subset}$ which, in particular, maps CB-derivations into functions of $\mathsf{B}^{\subset}$, thus concluding our characterisation of CB and CNB in terms of computational complexity.

7.1. The Translation Lemma.

A regular coderivation can be naturally seen as a finite tree with 'backpointers', a representation known as *cycle normal form*, cf. [Bro05, BS11].

Definition 53 (Cycle normal form)**.** Let $\mathcal{D}$ be a regular B^--coderivation. The *cycle normal form* (or simply *cycle nf*) of $\mathcal{D}$ is a pair $\langle \mathcal{D}, R_{\mathcal{D}}\rangle$, where $R_{\mathcal{D}}$ is a partial self-mapping on the nodes of $\mathcal{D}$ whose domain of definition is denoted $Bud(\mathcal{D})$ and:

(i) every infinite branch of $\mathcal{D}$ contains some (unique) $\nu \in Bud(\mathcal{D})$;
(ii) if $\nu \in Bud(\mathcal{D})$ then both $R_{\mathcal{D}}(\nu) \sqsubset \nu$ and $\mathcal{D}_{R_{\mathcal{D}}(\nu)} = \mathcal{D}_\nu$;
(iii) for any two distinct nodes $\mu \sqsubset \nu$ strictly below $Bud(\mathcal{D})$, $\mathcal{D}_\mu \neq \mathcal{D}_\nu$

We call any $\nu \in Bud(\mathcal{D})$ a *bud*, and $R_{\mathcal{D}}(\nu)$ its *companion*. A *terminal* node is either a leaf of $\mathcal{D}$ or a bud. The set of nodes of $\mathcal{D}$ bounded above by a terminal node is denoted $T_{\mathcal{D}}$. Given a node $\nu \in T_{\mathcal{D}}$, we define $Bud_\nu(\mathcal{D})$ as the restriction of buds to those above ν.

Remark 54. The cycle normal form of a regular coderivation $\mathcal{D}$ always exists, as by definition any infinite branch contains a node ν such that $\mathcal{D}_\nu = \mathcal{D}_\mu$ for some node μ below ν. $Bud(\mathcal{D})$ is designed to consist of just the *least* such nodes, so that by construction the cycle normal form is unique. Note that $Bud(\mathcal{D})$ must form an antichain: if $\mu, \nu \in Bud(\mathcal{D})$ with $\mu \sqsubset \nu$, then $R_{\mathcal{D}}(\mu) \sqsubset \mu$ are below $Bud(\mathcal{D})$ but we have $\mathcal{D}_{R_{\mathcal{D}}(\mu)} = \mathcal{D}_\mu$ by (ii) above, contradicting the (iii).

Also, notice that any branch of $\mathcal{D}$ contains a leaf of $T_{\mathcal{D}}$. Moreover, since $Bud(\mathcal{D})$ is an antichain, the leaves of $T_{\mathcal{D}}$ defines a 'bar' across $\mathcal{D}$, and so $T_{\mathcal{D}}$ is a finite tree.

The following proposition allows us to reformulate progressiveness, safety and left-leaning conditions for cycle normal forms.

Proposition 55. *Let $\mathcal{D}$ be a regular B^--coderivation with cycle nf $\langle \mathcal{D}, R_{\mathcal{D}}\rangle$. For any $\nu \in Bud(\mathcal{D})$, the (finite) path π from $R_{\mathcal{D}}(\nu)$ to ν satisfies:*

(1) if $\mathcal{D}$ is progressing, π must contain the conclusion of an instance of $\mathsf{cond}_{\Box N}$;

(2) *if $\mathcal{D}$ is a* CNB*-coderivation, π cannot contain the conclusion of* $\mathsf{cut}_{\Box N}$, $\Box_l$, $\mathsf{w}_\Box$, *and the leftmost premise of* $\mathsf{cond}_\Box$*;*
(3) *if $\mathcal{D}$ is a* CB*-coderivation, π cannot contain the conclusion of* w_N*, the leftmost premise of* cond_N*, and the rightmost premise of* cut_N*.*

In what follows we shall indicate circularities in cycle nfs explicitly by extending both CNB and CB with a new inference rule called dis:

$$\mathsf{dis}\,\frac{\Gamma \Rightarrow A}{\Gamma \Rightarrow A}\,X$$

where X is a finite set of nodes. In this presentation, we insist that each companion ν of the cycle nf $\langle \mathcal{D}, R_\mathcal{D}\rangle$ is always the conclusion of an instance of dis, where X denotes the set of buds ν' such that $R_\mathcal{D}(\nu') = \nu$. This expedient will allow us to formally distinguish cases when a node of $T_\mathcal{D}$ is a companion from those where it is the conclusion of a standard rule of B^-.

To facilitate the translation, we shall define two disjoint sets C_ν and O_ν. Intuitively, given a cycle nf $\langle \mathcal{D}, R_\mathcal{D}\rangle$ and $\nu \in T_\mathcal{D}$, C_ν is the set of companions above ν, while O_ν is the set of buds whose companion is strictly below ν.

Definition 56. Let $\langle \mathcal{D}, R_\mathcal{D}\rangle$ be the cycle nf of a B^--coderivation $\mathcal{D}$. We define the following two sets for any $\nu \in T_\mathcal{D}$:

$$C_\nu := \{\mu \in R_\mathcal{D}(Bud_\nu(\mathcal{D})) \mid \nu \sqsubseteq \mu\}$$
$$O_\nu := \{\mu \in Bud_\nu(\mathcal{D})) \mid R_\mathcal{D}(\mu) \sqsubset \nu\}$$

We are now ready to state and prove the Translation Lemma. Its proof proceeds by analysing each node $\nu_0 \in T_\mathcal{D}$ and associates with it an instance of the scheme $\mathsf{snrec}_\subset$ that simultaneously defines the functions $\{f_{\mathcal{D}_\nu} \mid \nu \in C_\nu \cup \{\nu_0\}\}$, with the help of an additional set of oracles $\{f_{\mathcal{D}_\mu} \mid \mu \in O_{\nu_0}\}$. When ν_0 is the root of $\mathcal{D}$, note that $O_{\nu_0} = \varnothing$ and so the function thus defined will be oracle-free. Thus we obtain an instance of $\mathsf{snrec}_\subset$ defining $f_\mathcal{D}$, and so $f_\mathcal{D} \in \mathsf{NB}^\subset$ by Proposition 42.

Lemma 57 (Translation Lemma)**.** *Let $\langle \mathcal{D}, R_\mathcal{D}\rangle$ be the cycle nf of a* CNB*-coderivation $\mathcal{D}$, and let $\nu_0 \in T_\mathcal{D}$:*

(1) *If $O_{\nu_0} = \varnothing$ then $f_{\mathcal{D}_{\nu_0}} \in \mathsf{NB}^\subset$. In particular, if $\mathcal{D}$ is a* CB*-coderivation then $f_{\mathcal{D}_{\nu_0}} \in \mathsf{B}^\subset$.*
(2) *If $O_{\nu_0} \neq \varnothing$ then $\forall \nu \in C_{\nu_0} \cup \{\nu_0\}$:*

$$f_{\mathcal{D}_\nu}(\vec{x};\vec{y}) = h_\nu((\lambda \vec{u} \subseteq \vec{x}, \lambda \vec{v}.f_{\mathcal{D}_\mu}(\vec{u};\vec{v}))_{\mu \in C_{\nu_0} \cup O_{\nu_0}})(\vec{x};\vec{y})$$

where:
(a) *$h_\nu \in \mathsf{NB}^\subset((f_{\mathcal{D}_\mu})_{\mu \in O_{\nu_0}}, (a_\mu)_{\mu \in C_{\nu_0}})$ and so $f_{\mathcal{D}_\nu} \in \mathsf{NB}^\subset((f_{\mathcal{D}_\mu})_{\mu \in O_{\nu_0}})$;*
(b) *the order $\vec{u} \subseteq \vec{x}$ is strict if either $\nu, \mu \in C_{\nu_0}$ or the path from ν to μ in $\mathcal{D}_{\nu_0}$ contains the conclusion of an instance of* $\mathsf{cond}_\Box$*;*
(c) *if $\mathcal{D}$ is a* CB*-coderivation then $\vec{v} \subseteq \vec{y}$.*

Proof sketch. Points 1 and 2 are proven by simultaneous induction on the longest distance of ν_0 from a leaf of $T_\mathcal{D}$. We just consider the most relevant case, i.e. when ν_0 is the conclusion of an instance of dis with premise ν', where X is the set of nodes labelling the rule. We have $O_{\nu_0} = O_{\nu'} \setminus X$ and $C_{\nu_0} = C_{\nu'} \cup \{\nu_0\}$. We want to find $(h_\nu)_{\nu \in C_\nu \cup \{\nu_0\}}$ defining the equations for $(f_{\mathcal{D}_\nu})_{\nu \in C_\nu \cup \{\nu_0\}}$ in such a way that points 2a-2c hold. We shall start by defining h_{ν_0}. First, note that, by definition of

cycle nf, $f_{\mathcal{D}_{\nu_0}}(\vec{x};\vec{y}) = f_{\mathcal{D}_{\nu'}}(\vec{x};\vec{y}) = f_{\mathcal{D}_\mu}(\vec{x};\vec{y})$ for all $\mu \in X$. By induction hypothesis on ν' there exists a family $(g_\nu)_{\nu \in C_{\nu'} \cup \{\nu'\}}$ such that:

$$f_{\mathcal{D}_{\nu'}}(\vec{x};\vec{y}) = g_{\nu'}((\lambda \vec{u} \subseteq \vec{x}, \lambda \vec{v}.f_{\mathcal{D}_\mu}(\vec{u};\vec{v}))_{\mu \in C_{\nu'} \cup O_{\nu'}})(\vec{x};\vec{y}) \tag{12}$$

and, moreover, for all $\nu \in C_{\nu'}$:

$$f_{\mathcal{D}_\nu}(\vec{x};\vec{y}) = g_\nu((\lambda \vec{u} \subseteq \vec{x}, \lambda \vec{v}.f_{\mathcal{D}_\mu}(\vec{u};\vec{v}))_{\mu \in C_{\nu'} \cup O_{\nu'}})(\vec{x};\vec{y}) \tag{13}$$

Since $O_{\nu'} = O_{\nu_0} \cup X$ and the path from ν' to any $\mu \in X$ must cross an instance of $\mathsf{cond}_\Box$ by Proposition 55.1, the induction hypothesis on ν' (point 2b) allows us to rewrite (12) as follows:

$$\begin{aligned} f_{\mathcal{D}_{\nu'}}(\vec{x};\vec{y}) = g_{\nu'}((&\lambda \vec{u} \subseteq \vec{x}, \lambda \vec{v}.f_{\mathcal{D}_\nu}(\vec{u};\vec{v}))_{\nu \in C_{\nu'}}, \\ &(\lambda \vec{u} \subseteq \vec{x}, \lambda \vec{v}.f_{\mathcal{D}_\mu}(\vec{u};\vec{v}))_{\mu \in O_{\nu_0}}, \\ &(\lambda \vec{u} \subset \vec{x}, \lambda \vec{v}.f_{\mathcal{D}_\mu}(\vec{u};\vec{v}))_{\mu \in X})(\vec{x};\vec{y}) \end{aligned} \tag{14}$$

On the other hand, for all $\nu \in \mathcal{C}_{\nu'}$, for all $\vec{u} \subseteq \vec{x}$ and $\vec{v}$, the equation in (13) can be rewritten as:

$$\begin{aligned} f_{\mathcal{D}_\nu}(\vec{u};\vec{v}) = g_\nu((&\lambda \vec{w} \subset \vec{u}, \lambda \vec{v'}.f_{\mathcal{D}_\mu}(\vec{u};\vec{v'}))_{\mu \in C_{\nu'}}, \\ &(\lambda \vec{w} \subseteq \vec{u}, \lambda \vec{v'}.f_{\mathcal{D}_\mu}(\vec{w};\vec{v'}))_{\mu \in O_{\nu_0}}, \\ &(\lambda \vec{w} \subseteq \vec{u}, \lambda \vec{v'}.f_{\mathcal{D}_\mu}(\vec{w};\vec{v'}))_{\mu \in X})(\vec{u};\vec{v}) \end{aligned} \tag{15}$$

and so, for all $\nu \in C_{\nu'}$:

$$\begin{aligned} \lambda \vec{u} \subseteq \vec{x}, \lambda v.f_{\mathcal{D}_\nu}(\vec{u};\vec{v}) = \lambda \vec{u} \subseteq \vec{x}, \lambda v.g_\nu((&\lambda \vec{w} \subset \vec{x}, \lambda \vec{v'}.f_{\mathcal{D}_\mu}(\vec{u};\vec{v'}))_{\mu \in C_{\nu'}}, \\ &(\lambda \vec{w} \subseteq \vec{x}, \lambda \vec{v'}.f_{\mathcal{D}_\mu}(\vec{w};\vec{v'}))_{\mu \in O_{\nu_0}}, \\ &(\lambda \vec{w} \subseteq \vec{x}, \lambda \vec{v'}.f_{\mathcal{D}_\mu}(\vec{w};\vec{v'}))_{\mu \in X})(\vec{u};\vec{v}) \end{aligned} \tag{16}$$

Now, since the paths from ν' to any $\mu \in X$ in $\mathcal{D}$ must contain an instance of $\mathsf{cond}_\Box$, for all $\nu \in C_{\nu'}$ and all $\mu \in X$, we have that either the path from ν' to ν contains an instance of $\mathsf{cond}_\Box$ or the path from ν to μ does. By applying the induction hypothesis on ν' (point 2b), given $\nu \in C_{\nu'}$ and $\mu \in X$, either $\lambda \vec{u} \subseteq \vec{x}, \lambda \vec{v}.f_{\mathcal{D}_\nu}(\vec{u};\vec{v})$ in (14) is such that $\vec{u} \subset \vec{x}$, or $\lambda \vec{w} \subseteq \vec{u}, \lambda \vec{v'}.f_{\mathcal{D}_\mu}(\vec{w};\vec{v'})$ in (15) is such that $\vec{w} \subset \vec{u}$. This means that, for any $\mu \in X$, $\lambda \vec{w} \subseteq \vec{x}, \lambda \vec{v'}.f_{\mathcal{D}_\mu}(\vec{w};\vec{v'})$ in (16) is such that $\vec{w} \subset \vec{x}$. For each $\nu \in C_{\nu'}$, by rewriting $\lambda \vec{u} \subseteq \vec{x}, \lambda \vec{v}.f_{\mathcal{D}_\nu}(\vec{u};\vec{v})$ in (14) according to the equation in (16) we obtain:

$$f_{\mathcal{D}_{\nu_0}}(\vec{x};\vec{y}) = t_{\nu_0}((\lambda \vec{u} \subset \vec{x}, \lambda \vec{v}.f_{\mathcal{D}_\mu}(\vec{u};\vec{v}))_{\mu \in C_{\nu'} \cup X}, (\lambda \vec{u} \subseteq \vec{x}, \lambda \vec{v}.f_{\mathcal{D}_\mu}(\vec{u};\vec{v}))_{\mu \in O_{\nu_0}})(\vec{x};\vec{y})$$

for some t_{ν_0}. Since $f_{\mathcal{D}_\mu} = f_{\mathcal{D}_{\nu_0}}$ for all $\mu \in X$, and since $C_{\nu_0} = C_{\nu'} \cup \{\nu_0\}$, by setting $h_{\nu_0} := t_{\nu_0}$ the above equation gives us the following:

$$f_{\mathcal{D}_{\nu_0}}(\vec{x};\vec{y}) = h_{\nu_0}((\lambda \vec{u} \subseteq \vec{x}, \lambda \vec{v}.f_{\mathcal{D}_\mu}(\vec{u};\vec{v}))_{\mu \in C_{\nu_0} \cup O_{\nu_0}})(\vec{x};\vec{y}) \tag{17}$$

which satisfies point 2b. From (17) we are able to find the functions $(h_\nu)_{\nu \in C_\nu}$ defining the equations for $(f_{\mathcal{D}_\nu})_{\nu \in C_\nu}$. Indeed, the induction hypothesis on ν' gives us (13) for any $\nu \in C_{\nu'}$. We rewrite in each such equation any $\lambda \vec{u} \subseteq \vec{x}, \lambda \vec{v}.f_{\mathcal{D}_\mu}(\vec{u};\vec{v})$ such that $\mu \in X$ according to equation (17), as $f_{\mathcal{D}_\mu} = f_{\mathcal{D}_{\nu_0}}$ for any $\mu \in X$. We obtain the following equation for any $\nu \in C_{\nu'}$:

$$f_{\mathcal{D}_\nu}(\vec{x};\vec{y}) = t_\nu((\lambda \vec{u} \subset \vec{x}, \lambda \vec{v}.f_{\mathcal{D}_\mu}(\vec{u};\vec{v}))_{\mu \in C_{\nu'} \cup \{\nu_0\}}, (\lambda \vec{u} \subseteq \vec{x}, \lambda \vec{v}.f_{\mathcal{D}_\mu}(\vec{u};\vec{v}))_{\mu \in O_{\nu_0}})(\vec{x};\vec{y})$$

for some t_ν. Since $C_{\nu_0} = C_{\nu'} \cup \{\nu_0\}$ and the above equation satisfies point 2b, we set $h_\nu := t_\nu$ and we obtain, for all $\nu \in C_{\nu_0}$:

$$f_{\mathcal{D}_\nu}(\vec{x};\vec{y}) = h_\nu((\lambda\vec{u} \subseteq \vec{x}, \lambda\vec{v}.f_{\mathcal{D}_\mu}(\vec{u};\vec{v}))_{\mu\in C_{\nu_0}\cup O_{\nu_0}})(\vec{x};\vec{y}) \tag{18}$$

It remains to show that (17) and (18) satisfy points 2a and 2c. Concerning point 2a, on the one hand for all $\nu \in C_{\nu_0}$ we have $h_\nu \in \mathsf{NB}^{\subset}((f_{\mathcal{D}_\mu})_{\mu\in O_{\nu_0}}, (a_\mu)_{\mu\in C_{\nu_0}})$, with $(a_\mu)_{\mu\in C_{\nu_0}}$ oracle functions. On the other hand, by applying the induction hypothesis, we have $f_{\mathcal{D}_{\nu_0}} = f_{\mathcal{D}_{\nu'}} \in \mathsf{NB}^{\subset}((f_{\mathcal{D}_\mu})_{\mu\in O_{\nu'}})$ and $f_{\mathcal{D}_\nu} \in \mathsf{NB}^{\subset}((f_{\mathcal{D}_\mu})_{\mu\in O_{\nu'}})$, for all $\nu \in C_{\nu_0}$. Since $O_{\nu'} = O_{\nu_0} \cup X$ and $f_{\mathcal{D}_{\nu_0}} = f_{\mathcal{D}_\mu}$ for all $\mu \in X$, we have both $f_{\mathcal{D}_{\nu_0}} \in \mathsf{NB}^{\subset}((f_{\mathcal{D}_\mu})_{\mu\in O_{\nu_0}})$ and $f_{\mathcal{D}_\nu} \in \mathsf{NB}^{\subset}((f_{\mathcal{D}_\mu})_{\mu\in O_{\nu_0}}, f_{\mathcal{D}_{\nu_0}})$ for all $\nu \in C_{\nu_0}$, and hence $f_{\mathcal{D}_\nu} \in \mathsf{NB}^{\subset}((f_{\mathcal{D}_\mu})_{\mu\in O_{\nu_0}})$, for all $\nu \in C_{\nu_0}$. Point 2c follows by applying the induction hypothesis, as the construction does not affect the safe arguments. □

Finally, we can establish the main result of this paper:

Corollary 58.

- $f(\vec{x};) \in \mathsf{CB}$ *iff* $f(\vec{x}) \in$ **FPTIME**;
- $f(\vec{x};) \in \mathsf{CNB}$ *iff* $f(\vec{x}) \in$ **FELEMENTARY**.

Proof. Follows directly from Theorem 49, Theorem 52 and Lemma 57. □

8. Conclusions

In this work we presented two-tiered circular type systems CB and CNB and showed that they capture polynomial-time and elementary computation, respectively. This is the first time that methods of circular proof theory have been applied in implicit computational complexity (ICC).

The widespread approach to ICC is based on the introduction of inductively defined languages or calculi endowed with recursion mechanisms whose strength is carefully calibrated in order to increase in complexity while not overstepping a given bound on computation. The circular proof systems CB and CNB pave the way to a radically different, top-down approach, where coinductive reasoning plays a central role.

We conclude this paper with some further remarks, some comments about expressivity, and some avenues for future work.

8.1. **Further remarks.** It is well-known that **FLINSPACE**, i.e. the class of functions computable in linear space, can be captured by reformulating B in *unary* notation (see [Bel92]). A similar result can be obtained for CB by just defining a unary version of the conditional in $\mathsf{B}^{\subset}$ (similarly to the ones in [Das21, KPP21b]) and by adapting the proofs of Lemma 43, Theorem 49 and Lemma 57. On the other hand, CNB is (unsurprisingly) not sensitive to such choice of notation.

Given the deep connection between the notion of regularity for coderivations and the notion of uniformity in computation discussed in Subsection 3.2, we may wonder if dropping the regularity condition from e.g. CB would result in a characterisation of the non-uniform version of **PTIME**, e.g. $\mathbf{P/poly}$. In other words, can we capture a non-uniform complexity class using the same proof-theoretic principles adopted to capture its uniform formulation? Unfortunately, the answer of this question is negative. To see this, notice that Example 59 shows how to define *any* number-theoretic function by a non-regular progressing B^{-}-coderivation, which happens to be both safe and left-leaning in a trivial way. It would nonetheless

be interesting if the regular/non-regular distinction in circular proof theory could somehow be leveraged to model the uniform/non-uniform distinction in computational complexity.

Let us finally observe that our proof-theoretic approach to 'circular complexity' was motivated by the inherently syntactical nature of threads, which define the progressing criterion, cf. Definition 14. Surprisingly enough, however, it seems that a purely term-oriented presentation of Corollary 58 could be possible thanks to Proposition 33, which shows that CB and CNB can be defined *independently* on the notion of thread. Consequently, our circular proof systems might be directly presented as 'function coalgebras' satisfying certain conditions or, alternatively, as 'two-sorted' Kleene-Herbrand-Gödel equational programs (as suggested by Definition 11). Such a presentation could be fruitful for future endeavours in this area.

8.2. **On intensional expressivity of circular systems.** Notice that Corollary 58 has an *extensional* flavour, as it refers only to usual function complexity classes. What about the *intensional* power of our circular proof systems? Can we gain some algorithmic expressivity by moving from a function algebra to its circular formulation? At least one way of comparing the intensional power of two-tiered systems in ICC is to compare their functions with both normal and safe inputs, not only normal inputs as is usually the case, cf. Corollary 58.

To this end, let us point out that the relations in Figure 1 between two-tiered systems indeed respect the safe-normal distinction of inputs. I.e. the inclusions $\mathsf{NB} \subseteq \mathsf{CNB} \subseteq \mathsf{NB}^{\subset}$ and $\mathsf{B} \subseteq \mathsf{CB} \subseteq \mathsf{B}^{\subset}$ indeed hold when construing each system as a set of safe-normal functions. In this way, note that showing $\mathsf{NB}^{\subset} \subseteq \mathsf{NB}$ or $\mathsf{B}^{\subset} \subseteq \mathsf{B}$ would be enough to infer that all systems have equivalent intensional expressivity, at the level of their definable safe-normal functions. We suspect that such a collapse does not hold, but we should point out that existing techniques for showing that a safe-normal function is not definable in a two-tiered system usually exploit a version of the Bounding Lemma from [BC92] (cf. Equation (7)), which all the systems considered here satisfy (cf. Lemma 43).

8.3. **On the 'power' of contraction.** Revisiting Remark 24, it not clear whether Corollary 58 can be restated in presence of the contraction rule $\mathsf{c}_{\square}$. Were this possible, NL-checkability of both CB and CNB would still hold as a consequence of Remark 35. Nonetheless, establishing soundness by means of a direct translation would be a rather difficult task, as the rule $\mathsf{c}_{\square}$ introduces more involved thread structures which invalidate some invariants of Lemma 57. Indeed in a setting where cut is context-splitting rather than context-sharing, it is known that contraction strictly increases extensional expressivity of regular progressing coderivations [KPP21b].

8.4. **Future work.** It would be pertinent to pursue higher-order versions of both CNB and CB, in light of precursory works in circular proof theory [Das21, KPP21b] as well as ICC [Hof97, Lei99, BNS00]. In the case of polynomial-time, for instance, a soundness result for some higher-order version of CB might follow by translation to (a sequent-style formulation of) Hofmann's SLR [Hof97]. Analogous translations might be defined for a higher-order version of CNB once the linearity restrictions on the recursion operator of SLR are dropped. Finally, as SLR is essentially a subsystem of Gödel's system T, such translations could refine the results on the abstraction complexity (i.e. type level) of the circular version of system T presented in [Das21].

We are also investigating how to extend the results of this paper to other relevant complexity classes, like **FPSPACE**. On the function algebraic side, notice that Lemma 43 implies a polyspace bound for functions in $\mathsf{SB}^{\subset}$, and hence for functions in SB. We conjecture that both function algebras capture precisely the class **FPSPACE**. Indeed, as already observed, the unnnested version of the recursion scheme snrec can be replaced by the scheme in (5), which allows both multiple recursive calls and composition during recursion. Several function algebras for **FPSPACE** have been proposed in the literature, and all of them involve variants of (5) (see [LM94, Oit08]).

These recursion schemes reflect the parallel nature of polynomial space functions, which in fact can be defined in terms of alternating polynomial time computation. We suspect that a circular proof theoretic characterisation of this class can be achieved by extending CB with the following parallel version of the cut rule:

$$\mathsf{pcut}\ \frac{\Gamma \Rightarrow N \quad \cdots \quad \Gamma \Rightarrow N \quad \Gamma, N, \ldots, N \Rightarrow N}{\Gamma \Rightarrow N}$$

and by adapting the left-leaning criterion in such a way as to allow some forms of composition during recursion. Indeed, we believe that SB itself characterises **FPSPACE**, but this result is beyond the scope of this work.

Parallel cut might also play a fundamental role for potential circular proof theoretic characterisations of circuit complexity classes, like **ALOGTIME** or **NC**. In such a setting, one would expect a conditional rule that implements a divide and conquer searching mechanism (see, e.g., [Lei98, LM00, BKMO16]).

References

[BC92] Stephen Bellantoni and Stephen Cook. A new recursion-theoretic characterization of the polytime functions (extended abstract). In *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing*, STOC '92, page 283–293, New York, NY, USA, 1992. Association for Computing Machinery.

[BDS16] David Baelde, Amina Doumane, and Alexis Saurin. Infinitary proof theory: the multiplicative additive case. 2016.

[Bel92] Stephen Bellantoni. *Predicative recursion and computational complexity*. Citeseer, 1992.

[BKMO16] Guillaume Bonfante, Reinhard Kahle, Jean-Yves Marion, and Isabel Oitavem. Two function algebras defining functions in nc^{k} boolean circuits. *Inf. Comput.*, 248:82–103, 2016.

[BNS00] Stephen J. Bellantoni, Karl-Heinz Niggl, and Helmut Schwichtenberg. Higher type recursion, ramification and polynomial time. *Ann. Pure Appl. Log.*, 104(1-3):17–30, 2000.

[Bro05] James Brotherston. Cyclic proofs for first-order logic with inductive definitions. In Bernhard Beckert, editor, *Automated Reasoning with Analytic Tableaux and Related Methods, International Conference, TABLEAUX 2005, Koblenz, Germany, September 14-17, 2005, Proceedings*, volume 3702 of *Lecture Notes in Computer Science*, pages 78–92. Springer, 2005.

[BS11] James Brotherston and Alex Simpson. Sequent calculi for induction and infinite descent. *Journal of Logic and Computation*, 21(6):1177–1216, 2011.

[Das18] Anupam Das. On the logical complexity of cyclic arithmetic. *arXiv preprint arXiv:1807.10248*, 2018.

[Das21] Anupam Das. On the logical strength of confluence and normalisation for cyclic proofs. In Naoki Kobayashi, editor, *6th International Conference on Formal Structures for Computation and Deduction, FSCD 2021, July 17-24, 2021, Buenos Aires, Argentina (Virtual Conference)*, volume 195 of *LIPIcs*, pages 29:1–29:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[DHL06a] Christian Dax, Martin Hofmann, and Martin Lange. A proof system for the linear time μ-calculus. In S. Arun-Kumar and Naveen Garg, editors, *FSTTCS 2006: Foundations of Software Technology and Theoretical Computer Science, 26th International Conference, Kolkata, India, December 13-15, 2006, Proceedings*, volume 4337 of *Lecture Notes in Computer Science*, pages 273–284. Springer, 2006.

[DHL06b] Christian Dax, Martin Hofmann, and Martin Lange. A proof system for the linear time μ-calculus. In *International Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 273–284. Springer, 2006.

[DP17] Anupam Das and Damien Pous. A cut-free cyclic proof system for kleene algebra. In *International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, pages 261–277. Springer, 2017.

[DP18] Anupam Das and Damien Pous. Non-Wellfounded Proof Theory For (Kleene+Action)(Algebras+Lattices). In Dan Ghica and Achim Jung, editors, *27th EACSL Annual Conference on Computer Science Logic (CSL 2018)*, volume 119 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 19:1–19:18, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[FS13] Jérôme Fortier and Luigi Santocanale. Cuts for circular proofs: semantics and cut-elimination. In *Computer Science Logic 2013 (CSL 2013)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.

[Hof97] Martin Hofmann. A mixed modal/linear lambda calculus with applications to bellantoni-cook safe recursion. In Mogens Nielsen and Wolfgang Thomas, editors, *Computer Science Logic, 11th International Workshop, CSL '97, Annual Conference of the EACSL, Aarhus, Denmark, August 23-29, 1997, Selected Papers*, volume 1414 of *Lecture Notes in Computer Science*, pages 275–294. Springer, 1997.

[Kle71] Stephen Cole Kleene. *Introduction to Metamathematics*. Bubliotheca Mathematica. Wolters-Noordhoff Publishing, 7 edition, 1971.

[KMPS19] Leszek Aleksander Kolodziejczyk, Henryk Michalewski, Pierre Pradic, and Michal Skrzypczak. The logical strength of büchi's decidability theorem. *Log. Methods Comput. Sci.*, 15(2), 2019.

[KPP19] Denis Kuperberg, Laureline Pinault, and Damien Pous. Cyclic proofs and jumping automata. In *FSTTCS*, 2019.

[KPP21a] Denis Kuperberg, Laureline Pinault, and Damien Pous. Cyclic proofs, system t, and the power of contraction. *Proc. ACM Program. Lang.*, 5(POPL):1–28, 2021.

[KPP21b] Denis Kuperberg, Laureline Pinault, and Damien Pous. Cyclic proofs, system t, and the power of contraction. *Proc. ACM Program. Lang.*, 5(POPL), January 2021.

[Lei91] Daniel Leivant. A foundational delineation of computational feasiblity. In *Proceedings of the Sixth Annual Symposium on Logic in Computer Science (LICS '91), Amsterdam, The Netherlands, July 15-18, 1991*, pages 2–11. IEEE Computer Society, 1991.

[Lei98] Daniel Leivant. A characterization of NC by tree recurrence. In *39th Annual Symposium on Foundations of Computer Science, FOCS '98, November 8-11, 1998, Palo Alto, California, USA*, pages 716–724. IEEE Computer Society, 1998.

[Lei99] Daniel Leivant. Ramified recurrence and computational complexity III: higher type recurrence and elementary complexity. *Ann. Pure Appl. Log.*, 96(1-3):209–229, 1999.

[LM94] Daniel Leivant and Jean-Yves Marion. Ramified recurrence and computational complexity II: substitution and poly-space. In Leszek Pacholski and Jerzy Tiuryn, editors, *Computer Science Logic, 8th International Workshop, CSL '94, Kazimierz, Poland, September 25-30, 1994, Selected Papers*, volume 933 of *Lecture Notes in Computer Science*, pages 486–500. Springer, 1994.

[LM00] Daniel Leivant and Jean-Yves Marion. A characterization of alternating log time by ramified recurrence. *Theor. Comput. Sci.*, 236(1-2):193–208, 2000.

[Min78] Grigori E Mints. Finite investigations of transfinite derivations. *Journal of Soviet Mathematics*, 10(4):548–596, 1978.

[NST19] Rémi Nollet, Alexis Saurin, and Christine Tasson. Pspace-completeness of a thread criterion for circular proofs in linear logic with least and greatest fixed points. In Serenella Cerrito and Andrei Popescu, editors, *Automated Reasoning with Analytic Tableaux and Related Methods - 28th International Conference, TABLEAUX 2019, London, UK, September 3-5, 2019, Proceedings*, volume 11714 of *Lecture Notes in Computer Science*, pages 317–334. Springer, 2019.

[NW96] Damian Niwiński and Igor Walukiewicz. Games for the μ-calculus. *Theoretical Computer Science*, 163(1-2):99–116, 1996.
[Oit08] Isabel Oitavem. Characterizing PSPACE with pointers. *Math. Log. Q.*, 54(3):323–329, 2008.
[Ros84] H. E. Rose. *Subrecursion: Functions and Hierarchies*. Oxford University Press, 1984.
[WW99] Marc Wirz and Marc Wirz. Characterizing the grzegorczyk hierarchy by safe recursion, 1999.

Appendix A. Proofs of Section 2

Proof of Proposition 5. The proof is by induction on the size of the derivation. The case where the last rule of $\mathcal{D}$ is an instance of $\mathsf{id}, 0, \Box_r, \mathsf{cond}_N, \mathsf{cond}_\Box$, or srec are trivial. If the last rule of $\mathcal{D}$ is an instance of $\mathsf{e}_N, \mathsf{e}_\Box, \Box_l, \mathsf{s}_i$, and $\mathsf{w}_\Box$ then we apply the induction hypothesis. Let us now suppose that $\mathcal{D}$ has been obtained from a derivation $\mathcal{D}_0$ by applying an instance of w_N. By induction hypothesis, there exists a derivation $\mathcal{D}_0^* : \Box N, \overset{n}{\ldots}, \Box N \Rightarrow \Box N$ such that $f_{\mathcal{D}_0}(\vec{x};\vec{y}) = f_{\mathcal{D}_0^*}(\vec{x};)$. Since $f_{\mathcal{D}}(\vec{x};\vec{y},y) = f_{\mathcal{D}_0}(\vec{x};\vec{y}) = f_{\mathcal{D}_0^*}(\vec{x};)$ we just set $\mathcal{D}^* = \mathcal{D}_0^*$. Suppose now that $\mathcal{D}$ is obtained from two derivations $\mathcal{D}_0$ and $\mathcal{D}_1$ by applying an instance of cut_N. By induction hypothesis, there exists $\mathcal{D}_1^*$ such that $f_{\mathcal{D}_1}(\vec{x};\vec{y},y) = f_{\mathcal{D}_1^*}(\vec{x};)$. Since $f_{\mathcal{D}_1}(\vec{x};\vec{y}, f_{\mathcal{D}_0}(\vec{x};\vec{y})) = f_{\mathcal{D}_1^*}(\vec{x};)$, we set $\mathcal{D}^* = \mathcal{D}_1^*$. As for the case where the last rule is $\mathsf{cut}_\Box$, by induction hypothesis, there exist derivations $\mathcal{D}_0^*$ and $\mathcal{D}_1^*$ such that $f_{\mathcal{D}_0}(\vec{x};\vec{y}) = f_{\mathcal{D}_0^*}(\vec{x};)$ and $f_{\mathcal{D}_1}(\vec{x},x;\vec{y}) = f_{\mathcal{D}_1^*}(\vec{x},x;)$, so that we define $\mathcal{D}^*$ as the derivation obtained from $\mathcal{D}_0^*$ and $\mathcal{D}_1^*$ by applying the rule $\mathsf{cut}_\Box$. □

Proof of Proposition 6. Let us first prove the left-right implication by induction on $f \in \mathsf{B}$. The cases where f is 0 or s_i are straightforward. If f is a projection then we construct $\mathcal{D}$ using the rules id, w_N, $\mathsf{w}_\Box$, e_N, and $\mathsf{e}_\Box$. If $f = \mathsf{p}(;x)$ then $\mathcal{D}$ is as follows:

$$\mathsf{cond}_N \dfrac{\overset{0}{\dfrac{}{\Rightarrow N}} \quad \overset{\mathsf{id}}{\dfrac{}{N \Rightarrow N}} \quad \overset{\mathsf{id}}{\dfrac{}{N \Rightarrow N}}}{N \Rightarrow N}$$

If f is $\mathsf{cond}(;x,y,z,w)$ then $\mathcal{D}$ is constructed using id, w_N, e_N and cond_N. Suppose now that $f(\vec{x};\vec{y}) = h(\vec{x}, g(\vec{x};);\vec{y})$. Then $\mathcal{D}$ is as follows:

$$\mathsf{cut}_\Box \dfrac{\Box_r \dfrac{\overset{\mathcal{D}_0}{\Box\vec{N} \Rightarrow N}}{\Box\vec{N} \Rightarrow \Box N} \qquad \overset{\mathcal{D}_1}{\Box\vec{N}, \Box N, \vec{N} \Rightarrow N}}{\Box\vec{N}, \vec{N} \Rightarrow N}$$

where $\mathcal{D}_0$ and $\mathcal{D}_1$ are such that $f_{\mathcal{D}_0} = g$ and $f_{\mathcal{D}_1} = h$. If $f(\vec{x};\vec{y}) = h(\vec{x};\vec{y}, g(\vec{x};\vec{y}))$, then $\mathcal{D}$ is as follows:

$$\mathsf{cut}_N \dfrac{\overset{\mathcal{D}_0}{\Box\vec{N}, \vec{N} \Rightarrow N} \qquad \overset{\mathcal{D}_1}{\Box\vec{N}, \vec{N}, N \Rightarrow N}}{\Box\vec{N}, \vec{N} \Rightarrow N}$$

where $\mathcal{D}_0$ and $\mathcal{D}_1$ are such that $f_{\mathcal{D}_0} = g$ and $f_{\mathcal{D}_1} = h$. Last, suppose that $f(x,\vec{x};\vec{y})$ has been obtained by safe recursion from $g(\vec{x};\vec{y})$ and $h_i(x,\vec{x};\vec{y},y)$ with $i = 0, 1$.

Then, $\mathcal{D}$ is as follows:

$$\mathsf{srec}\ \dfrac{\overset{\mathcal{D}_0}{\square\vec{N},\vec{N}\Rightarrow N}\quad \overset{\mathcal{D}_1}{\square N,\square\vec{N},\vec{N},N\Rightarrow N}\quad \overset{\mathcal{D}_2}{\square N,\square\vec{N},\vec{N},N\Rightarrow N}}{\square N,\square\vec{N},\vec{N}\Rightarrow N}$$

where $\mathcal{D}_0$, $\mathcal{D}_1$, and $\mathcal{D}_2$ are such that $f_{\mathcal{D}_0}=g$, $f_{\mathcal{D}_1}=h_0$ and $f_{\mathcal{D}_2}=h_1$.

For the right-left implication, we prove by induction on the size of derivations that $\mathcal{D}:\square N,\overset{n}{\ldots},\square N,N,\overset{m}{\ldots},N\Rightarrow C$ implies $f_{\mathcal{D}}\in\mathsf{B}$. The cases where the last rule of $\mathcal{D}$ is $\mathsf{id},0,\mathsf{s}_i,\mathsf{w}_N,\mathsf{w}_\square,\mathsf{e}_N,\mathsf{e}_\square,\square_l,\square_r$ are all straightforward using constants, successors and projections. If $\mathcal{D}$ has been obtained from two derivations $\mathcal{D}_0$ and $\mathcal{D}_1$ by applying an instance of cut_N then, by applying the induction hypothesis, $f_{\mathcal{D}}(\vec{x};\vec{y})=f_{\mathcal{D}_0}(\vec{x};\vec{y},f_{\mathcal{D}_1}(\vec{x};\vec{y}))\in\mathsf{B}$. As for the case where the last rule is $\mathsf{cut}_\square$, by Proposition 5 there exists a derivation $\mathcal{D}_1^*$ with smaller size such that $f_{\mathcal{D}_1}(\vec{x};\vec{y})=f_{\mathcal{D}_1^*}(\vec{x};)$. By applying the induction hypothesis, we have $f_{\mathcal{D}}(\vec{x};\vec{y})=f_{\mathcal{D}_0}(\vec{x},f_{\mathcal{D}_1^*}(\vec{x};);\vec{y})\in\mathsf{B}$. If $\mathcal{D}$ has been obtained from derivations $\mathcal{D}_0$, $\mathcal{D}_1$, $\mathcal{D}_2$ by applying an instance of cond_N then, by using the induction hypothesis we have $f_{\mathcal{D}}(\vec{x};\vec{y},y)=\mathsf{cond}(;y,f_{\mathcal{D}_0}(\vec{x};\vec{y}),f_{\mathcal{D}_1}(\vec{x};\vec{y},\mathsf{p}(;y)),f_{\mathcal{D}_2}(\vec{x};\vec{y},\mathsf{p}(;y)))\in\mathsf{B}$. As for the case where the last rule is $\mathsf{cond}_\square$, by applying the induction hypothesis we have $f_{\mathcal{D}}(x,\vec{x};\vec{y})=\mathsf{cond}(;x,f_{\mathcal{D}_0}(\vec{x};\vec{y}),f_{\mathcal{D}_1}(\mathsf{p}(x;),\vec{x};\vec{y}),f_{\mathcal{D}_2}(\mathsf{p}(x;),\vec{x};\vec{y}))\in\mathsf{B}$, where $\mathsf{p}(x;)=\mathsf{p}(;\pi^{1;0}_{1;}(x;))$. Last, if $\mathcal{D}$ has been obtained from derivations $\mathcal{D}_0$, $\mathcal{D}_1$, $\mathcal{D}_2$ by applying an instance of srec, then $f_{\mathcal{D}}\in\mathsf{B}$ using the induction hypothesis and the safe recursion scheme. □

Appendix B. Further proofs and examples for Section 3

B.1. **Examples for Proposition 21.** In what follows, in light of Proposition 20, we shall simply omit modalities in (regular) (progressing) coderivations, i.e. we shall regard any formula in the context of a sequent as modal and we shall omit applications of $\square_r$. Consequently, we shall write e.g. cut instead of $\mathsf{cut}_{\square N}$ and avoid writing semicolons in the semantics of a coderivation.

Following essentially [Das21], the first fundamental observation is that any type 1 function is B^--definable by a progressing coderivation.

Example 59 (Extensional completeness at type 1)**.** For any function $f:\mathbb{N}^k\to\mathbb{N}$ there is a progressing coderivation $\mathcal{F}$ such that $f_{\mathcal{F}}=f$. Proceeding by induction on k, if $k=0$ then we use the rules $0,\mathsf{s}_0,\mathsf{s}_1,\mathsf{cut}$ to construct $\mathcal{F}$ defining the natural number f. Otherwise, suppose $f:\mathbb{N}\times\mathbb{N}^k\to\mathbb{N}$ and define f_n as $f_n(\vec{x})=f(n,\vec{x})$. We construct the coderivation defining f as follows:

$$\mathsf{cut}\ \dfrac{\overset{\mathcal{N}}{N,\vec{N}\Rightarrow N}\quad \mathsf{cond}\ \dfrac{\overset{f_0}{\vec{N}\Rightarrow N}\qquad \mathsf{cond}\ \dfrac{\overset{f_1}{\vec{N}\Rightarrow N}\qquad \mathsf{cond}\ \dfrac{\overset{f_2}{\vec{N}\Rightarrow N}\qquad \mathsf{cond}\ \dfrac{\vdots}{N,\vec{N}\Rightarrow N}}{N,\vec{N}\Rightarrow N}}{N,\vec{N}\Rightarrow N}}{N,\vec{N},\Rightarrow N}}{N,\vec{N}\Rightarrow N}$$

where $\mathcal{N}$ is the coderivation converting n into $\mathsf{s}_1.^n.\mathsf{s}_1 0$ (see Appendix E), and we omit the second premise of cond as it is never selected. Notice that, by a cardinality argument, $\mathcal{F}$ is not regular in general: there are only countable many regular coderivations, while there are continuum many functions over $\mathbb{N}$. Moreover, $\mathcal{F}$ is progressing, as red formulas N (which are modal) form a progressing thread.

The above example illustrates the role of regularity as a *uniformity* condition: regular coderivations admit a finite description (e.g. a finite tree with backpointers), so that they define computable functions. In fact, it turns out that any (partial) recursive function is B^--definable by a regular coderivation. This can be easily inferred from the next two examples using Proposition 20.

Example 60 (Primitive recursion)**.** Let $f(x,\vec{x})$ be defined by primitive recursion from $g(\vec{x})$ and $h(x,\vec{x},y)$. Given coderivations $\mathcal{G}$ and $\mathcal{H}$ defining g and h respectively, we construct the following coderivation $\mathcal{R}$:

$$\dfrac{\mathcal{N}\ \ N \Rightarrow N \qquad \dfrac{\mathcal{G}\ \ \vec{N} \Rightarrow N \qquad \dfrac{\mathcal{P}\ \ N \Rightarrow N \qquad \dfrac{\vdots}{\mathsf{cond}\ \dfrac{N, N, \vec{N} \Rightarrow N}{\bullet}} \quad N, \vec{N}, N \Rightarrow N}{\mathsf{cut}\ \dfrac{N, N, \vec{N} \Rightarrow N}{\bullet}}}{\mathsf{cond}\ \dfrac{\underline{N}, N, \vec{N} \Rightarrow N}{\bullet}}}{\mathsf{cut}\ \dfrac{N, \vec{N} \Rightarrow N}{}}$$

where $\mathcal{N}$ is the coderivation converting n into $\mathsf{s}_1.^n.\mathsf{s}_1 0$, $\mathcal{P}$ is the coderivation defining unary predecessor (see Appendix E), and we avoid writing the second premise of cond as it is never selected. Notice that $\mathcal{R}$ is progressing, as blue formulas N (which are modal) form a progressing thread contained in the infinite branch that loops on $\bullet$. From the associated equational program we obtain:

$$\begin{array}{rcl} f_{\mathcal{R}_\epsilon}(x,\vec{x}) & = & f_{\mathcal{R}1}(f_{\mathcal{N}}(x), x, \vec{x}) \\ f_{\mathcal{R}_1}(0,x,\vec{x}) & = & g(\vec{x}) \\ f_{\mathcal{R}_1}(\mathsf{s}_1 z, x, \vec{x}) & = & h(f_{\mathcal{P}}(x), \vec{x}, f_{\mathcal{R}_1}(z, f_{\mathcal{P}}(x), \vec{x})) \end{array}$$

so that $f_{\mathcal{R}} = f_{\mathcal{R}\epsilon} = f$.

Example 61 (Unbounded search)**.** Let $g(x,\vec{x})$ be a function, and let $f(\vec{x}) := \mu x.(g(x,\vec{x}) = 0)$ be the unbounded search function obtained by applying the minimisation operation on $g(\vec{x})$. Given a coderivation $\mathcal{G}$ defining g, we construct the following coderivation $\mathcal{U}$:

$$\dfrac{\dfrac{0}{\Rightarrow N}\quad \dfrac{\mathcal{G}\ \ N, \vec{N} \Rightarrow N \qquad \dfrac{\mathsf{id}\ \dfrac{}{N \Rightarrow N} \qquad \dfrac{\mathcal{S}\ \ N \Rightarrow N \qquad \dfrac{\vdots}{\mathsf{cut}\ \dfrac{N, \vec{N} \Rightarrow N}{\bullet}}}{\mathsf{cut}\ \dfrac{N, \vec{N} \Rightarrow N}{}}}{\mathsf{cond}\ \dfrac{\underline{N}, N, \vec{N} \Rightarrow N}{\bullet}}}{\mathsf{cut}\ \dfrac{N, \vec{N} \Rightarrow N}{}}}{\mathsf{cut}\ \dfrac{\vec{N} \Rightarrow N}{}}$$

where the coderivation $\mathcal{N}$ computes the unary successor (see Example 10), and we identify the sub-coderivations corresponding to the second and the third premises of the conditional rule. It is easy to check that the above coderivation is regular but not progressing, as threads containing principal formulas for cond are finite. From the associated equational program we obtain:

$$\begin{array}{rcll} f_{\mathcal{U}_\epsilon}(\vec{x}) & = & f_{\mathcal{U}_1}(0,\vec{x}) & \\ f_{\mathcal{U}_1}(x,\vec{x}) & = & f_{\mathcal{U}_{11}}(f_{\mathcal{G}}(x,\vec{x}),x,\vec{x}) & \\ f_{\mathcal{U}_{11}}(0,x,\vec{x}) & = & x & \\ f_{\mathcal{U}_{11}}(\mathsf{s}_0 z,x,\vec{x}) & = & f_{\mathcal{U}_1}(f_{\mathcal{S}}(x),\vec{x}) & z \neq 0 \\ f_{\mathcal{U}_{11}}(\mathsf{s}_1 z,x,\vec{x}) & = & f_{\mathcal{U}_1}(f_{\mathcal{S}}(x),\vec{x}) & \end{array}$$

Which searches for the least $x \geq 0$ such that $g(x,\vec{x}) = 0$. Hence, $f_{\mathcal{U}_\epsilon}(\vec{x}) = f_{\mathcal{U}}(\vec{x}) = f(\vec{x})$.

B.2. **Other proofs for Section 3.**

Proof of Theorem 22. First, by Proposition 20 we can neglect modalities in B^--coderivations (and semicolons in the corresponding semantics). The left-right implication thus follows from the natural inclusion of our system into CT_0 and [Das21, Corollary 80].

Concerning the right-left implication, we employ a formulation $\mathsf{T}_1(\vec{a})$ of the type 1 functions of T_1 over oracles $\vec{a}$ as follows. $\mathsf{T}_1(\vec{a})$ is defined just like the primitive recursive functions, including oracles $\vec{a}$ as initial functions, and by adding the following version of type 1 recursion:

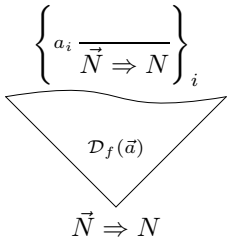
- if $g(\vec{x}) \in \mathsf{T}_1(\vec{a})$ and $h(a)(x,\vec{x}) \in \mathsf{T}_1(a,\vec{a})$, then the $f(x,\vec{x})$ given by,

$$\begin{array}{rcl} f(0,\vec{x}) & = & g(\vec{x}) \\ f(\mathsf{s}_i x,\vec{x}) & = & h_i(\lambda\vec{u}.f(x,\vec{u}))(x,\vec{x}) \end{array}$$

  is in $\mathsf{T}_1(\vec{a})$.

It is not hard to see that the type 1 functions of T_1 are precisely those of $\mathsf{T}_1(\varnothing)$. We then conclude by showing how to define the above scheme by regular and progressing B^--coderivations.

Given a function $f(\vec{x}) \in \mathsf{T}_1(\vec{a})$, we construct a regular progressing coderivation of B^-,



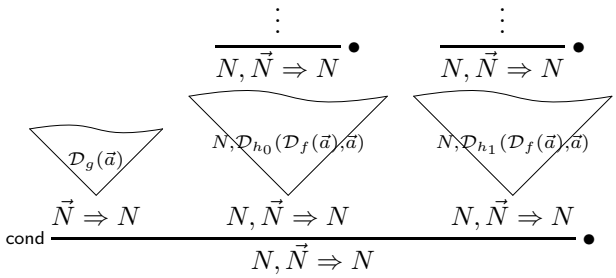
computing $f(\vec{x})$ over $\vec{a}$, by induction on the definition of $f(\vec{x})$.

If $f(\vec{x})$ is an initial function, an oracle, or $f(\vec{x}) = h(g(\vec{x}),\vec{x})$ then the construction is easy (see, e.g., the proof of Theorem 52).

Let us consider the case of recursion (which subsumes usual primitive recursion at type 0). Suppose $f(x,\vec{x}) \in \mathsf{T}_1(\vec{a})$ where,

$$\begin{array}{rcl} f(0,\vec{x}) & = & g(\vec{x}) \\ f(\mathsf{s}_0 x,\vec{x}) & = & h_0(\lambda\vec{u}.f(x,\vec{u}))(x,\vec{x}) \\ f(\mathsf{s}_1 x,\vec{x}) & = & h_1(\lambda\vec{u}.f(x,\vec{u}))(x,\vec{x}) \end{array}$$

where $\mathcal{D}_g(\vec{a})$ and $\mathcal{D}_h(a, \vec{a})$ are already obtained by the inductive hypothesis. We define $\mathcal{D}_f(\vec{a})$ as follows:



Note that the existence of the second and third coderivations above the conditional is given by a similar construction to that of Lemma 51 (see also [Das21, Lemma 42]). □

B.3. **Argument for Remark 35.**

We show that any safe B^--coderivation with contraction rules is progressing whenever its infinite branches cross infinitely many $\mathsf{cond}_\Box$-steps. First, notice that due to explicit contraction there can be distinct threads contained in a branch B that start with the same modal formula. For this reason, instead of considering threads, we shall consider graphs of immediate ancestry (see Definition 13 and Definition 14), which collect many threads. Given an infinite branch B, by safety there exists a node ν of B such that any sequent above ν is not the conclusion of a $\mathsf{cut}_\Box$-step. Now, by inspecting the rules of $\mathsf{B}^- \setminus \{\mathsf{cut}_\Box\}$ we observe that:

- given a modal formula in the context of the sequent at ν there is a unique ancestry tree in B that starts at ν and contains that modal formula;
- any ancestry tree in B starting from a node above ν can be extended to a one starting at ν.

Hence, B contains k ancestry trees $T_1, \ldots, T_k$, where k is the number of modal formulas in the context of the sequent at ν. Now, any such ancestry tree T_i induces a tree T_i' such that:

- a node T_i' is either the root of T_i or a node of T_i that is principal formula of a $\mathsf{cond}_\Box$-step.
- an edge of T_i' from ν to μ exists if there is a path in T_i from ν to μ.

each T_i' is clearly finitely branching. Since B contains infinitely many $\mathsf{cond}_\Box$-steps, by the Infinite Pigeonhole Principle we conclude that some $i \leq k$ exists such that T_i' is infinite. By König lemma we conclude that T_i' contains an infinite branch, and so T_i contains a thread crossing infinitely many $\mathsf{cond}_\Box$-steps. This shows that any infinite branch contains a progressing thread.

Appendix C. Proofs of Section 6

In this section we prove Corollary 48. To begin with, we recall the definition of the class **FELEMENTARY**:

Definition 62. FELEMENTARY is the smallest set of functions containing:

- $0() := 0 \in \mathbb{N}$,
- $\pi_i^n(x_1, \ldots, x_n) := x_j$, whenever $1 \leq j \leq n$;
- $\mathsf{s}(x) := x + 1$;

- the function E_2 defined as follows:

$$E_1(x) = x^2 + 2$$
$$E_2(0) = 2$$
$$E_2(x+1) = E_1(E_2(x))$$

and closed under the following:

- (Composition) If $f(\vec{x}, x), g(\vec{x}) \in \mathbf{FELEMENTARY}$ then so is $f(\vec{x}, g(\vec{x}))$;
- (Bounded recursion) If $g(\vec{x}), h(x, \vec{x}, y), j(x, \vec{x}) \in \mathbf{FELEMENTARY}$ then so is $f(x, \vec{x})$ given by:

$$f(0, \vec{x}) := g(\vec{x})$$
$$f(x+1, \vec{x}) := h(x, \vec{x}, f(x, \vec{x}))$$

provided that $f(x, \vec{x}) \leq j(x, \vec{x})$.

Proposition 63 ([Ros84])**.** *Let $f \in \mathbf{FELEMENTARY}$ be a k-ary function. Then, there exists an integer m such that:*

$$f(\vec{x}) \leq E_2^m(\max_k(\vec{x}))$$

where $E_2^0(x) = x$ and $E_2^{m+1}(x) = E_2(E_2^m(x))$.

For our purposes we shall consider a formulation of this class in binary notation, that we call $\mathbf{FELEMENTARY}_{0,1}$.

Definition 64. $\mathbf{FELEMENTARY}_{0,1}$ is the smallest set of functions containing:

- $0() := 0 \in \mathbb{N}$,
- $\pi_i^n(x_1, \ldots, x_n) := x_j$, whenever $1 \leq j \leq n$;
- $\mathsf{s}_i(x) := 2x + i$, for $i \in \{0, 1\}$
- the function $\varepsilon(x, y)$ defined as follows:

$$\varepsilon(0, y) := \mathsf{s}_0(y)$$
$$\varepsilon(\mathsf{s}_i x, y) := \varepsilon(x, \varepsilon(x, y))$$

and closed under the following:

- (Composition) If $f(\vec{x}, x), g(\vec{x}) \in \mathbf{FELEMENTARY}_{0,1}$ then so is $f(\vec{x}, g(\vec{x}))$;
- (Bounded recursion on notation) If $g(\vec{x}), h_0(x, \vec{x}, y), h_1(x, \vec{x}, y), j(x, \vec{x}) \in \mathbf{FELEMENTARY}_{0,1}$ then so is $f(x, \vec{x})$ given by:

$$f(0, \vec{x}) := g(\vec{x})$$
$$f(\mathsf{s}_i x, \vec{x}) := h_i(x, \vec{x}, f(x, \vec{x}))$$

provided that $f(x, \vec{x}) \leq j(x, \vec{x})$.

It is easy to show that the unary and the binary definition of the class of elementary time computable functions coincide. To see this, we first define $\varepsilon^n(x)$ as

$$\begin{array}{rcl} \varepsilon^1(x) & := & \varepsilon(x, 1) \\ \varepsilon^{m+1}(x) & := & \varepsilon^1(\varepsilon^m(x)) \end{array} \tag{19}$$

which allows us to prove that $\varepsilon^n(x)$ plays the role of rate growth function as the function $E_2^n(x)$ (see Proposition 63).

Proposition 65.

(1) $\mathbf{FELEMENTARY} = \mathbf{FELEMENTARY}_{0,1}$*;*

*(2) for any $f \in$ **FELEMENTARY**$_{0,1}$ k-ary function there is an integer m such that:*

$$f(\vec{x}) \leq \varepsilon^m(\max_k(\vec{x}))$$

Proof. Let us first prove point 1. For the $\supseteq$ direction we show that for any $f \in$ **FELEMENTARY**$_{0,1}$ there exists n such that:

$$|f(\vec{x})| \leq 2_n(\sum |\vec{x}|) \tag{20}$$

where $2_0(x) = x$ and $2_{n+1}(x) = 2^{2_n(x)}$. Since $|x| = \lceil \log_2(x+1) \rceil$, from the above inequation we would have that, for some m:

$$f(\vec{x}) \leq 2_{n+m}(\sum \vec{x})$$

which allows us to conclude $f \in$ **FELEMENTARY**, as the elementary time computable functions are exactly the elementary space ones. The inequation (20) can be proved by induction on f, noticing that $|\varepsilon(x,y)| = 2^{|x|} + |y|$. Concerning the $\subseteq$ direction, we prove by induction on $f \in$ **FELEMENTARY** that there exists a function $\hat{f} \in$ **FELEMENTARY**$_{0,1}$ such that, for all $\vec{x} = x_1, \ldots, x_n$:

$$f(\vec{x}) = |\hat{f}(\mathsf{s}_1^{x_1}(0), \ldots, \mathsf{s}_1^{x_n}(0))|$$

where $\mathsf{s}_1^m(0) = \mathsf{s}_1(\overset{m}{\ldots}\mathsf{s}_1(0))$. Since the functions $|\cdot|$ and $x \mapsto \mathsf{s}_1^x(0)$ are both in **FELEMENTARY**$_{0,1}$, we are able to conclude $f \in$ **FELEMENTARY**$_{0,1}$. The case $f = 0$ is trivial. As for the cases $f = \mathsf{s}$ and $f = \pi_i^n$, we first notice that $|\mathsf{s}_1^x(0)| = x$. Then, we have:

$$\begin{aligned} \mathsf{s}(x) &= |\mathsf{s}_1^{\mathsf{s}(x)}(0)| \\ \pi_i^n(\vec{x}) = x_i &= |\mathsf{s}_1^{x_i}(0)| = |\pi_i^n(\mathsf{s}_1^{x_1}(0), \ldots, \mathsf{s}_1^{x_n}(0))| \end{aligned}$$

Concerning the case of $E_2(x)$, we first notice that the following property holds for any m and some k:

$$E_2^m(x) \leq \varepsilon^{m+k}(x) \tag{21}$$

where $\varepsilon^n(x)$ is as in (19). Moreover, the function $E_2(x)$ can be defined by two applications of bounded recursion proceeding from the successor and the projection functions, where each recursion can be bounded by $E_2(x)$, and hence by $\varepsilon^k(x)$ for some k. This means that the case of $E_2(x)$ can be reduced to the case of bounded recursion. Suppose now that $f(\vec{x}) = h(\vec{x}, g(\vec{x}))$. We define $\hat{f}(\vec{x}) = \hat{h}(\vec{x}, \hat{g}(\vec{x}))$ so that, by induction hypothesis:

$$\begin{aligned} f(\vec{x}) &= h(\vec{x}, g(\vec{x})) \\ &= |\hat{h}(\mathsf{s}_1^{x_1}(0), \ldots, \mathsf{s}_1^{x_n}(0), \mathsf{s}_1^{g(\vec{x})}(0))| \\ &= |\hat{h}(\mathsf{s}_1^{x_1}(0), \ldots, \mathsf{s}_1^{x_n}(0), \mathsf{s}_1^{|\hat{g}(\mathsf{s}_1^{x_1}(0), \ldots, \mathsf{s}_1^{x_n}(0))|}(0))| \\ &= |\hat{h}(\mathsf{s}_1^{x_1}(0), \ldots, \mathsf{s}_1^{x_n}(0), \hat{g}(\mathsf{s}_1^{x_1}(0), \ldots, \mathsf{s}_1^{x_n}(0)))| \\ &= |\hat{f}(\vec{x})| = |\hat{h}(\vec{x}, \hat{g}(\vec{x}))| \end{aligned}$$

Last, suppose that f has been obtained by bounded recursion from h, g, j, i.e.

$$\begin{aligned} f(0, \vec{x}) &= g(\vec{x}) \\ f(y+1, \vec{x}) &= h(y, \vec{x}, f(y, \vec{x})) \end{aligned}$$

provided that $f(y,\vec{x}) \leq j(y,\vec{x})$. We define $\hat{f}$ as follows:

$$\hat{f}(0,\vec{x}) = \hat{g}(\vec{x})$$
$$\hat{f}(\mathsf{s}_i y,\vec{x}) = \hat{h}(y,\vec{x},\hat{f}(y,\vec{x}))$$

We show by induction on y that:

$$f(y,\vec{x}) = |\hat{f}(\mathsf{s}_1^y(0),\mathsf{s}_1^{x_1}(0),\ldots,\mathsf{s}_1^{x_n}(0))|$$

We have:

$$f(0,\vec{x}) = g(\vec{x}) = |\hat{g}(\mathsf{s}_1^{x_1}(0),\ldots,\mathsf{s}_1^{x_n}(0))| = |\hat{f}(\mathsf{s}_1^0(0),\mathsf{s}_1^{x_1}(0),\ldots,\mathsf{s}_1^{x_n}(0))|$$

$$\begin{aligned}
f(y+1,\vec{x}) &= h(y,\vec{x},f(y,\vec{x})) \\
&= |\hat{h}(\mathsf{s}_1^y(0),\mathsf{s}_1^{x_1}(0),\ldots,\mathsf{s}_1^{x_n}(0),\mathsf{s}_1^{f(y,\vec{x})}(0))| \\
&= |\hat{h}(\mathsf{s}_1^y(0),\mathsf{s}_1^{x_1}(0),\ldots,\mathsf{s}_1^{x_n}(0),\mathsf{s}_1^{|\hat{f}(\mathsf{s}_1^y(0),\mathsf{s}_1^{x_1}(0),\ldots,\mathsf{s}_1^{x_n}(0))|}(0))| \\
&= |\hat{h}(\mathsf{s}_1^y(0),\mathsf{s}_1^{x_1}(0),\ldots,\mathsf{s}_1^{x_n}(0),\hat{f}(\mathsf{s}_1^y(0),\mathsf{s}_1^{x_1}(0),\ldots,\mathsf{s}_1^{x_n}(0))| \\
&= |\hat{f}(\mathsf{s}_1^{y+1}(0),\mathsf{s}_1^{x_1}(0),\ldots,\mathsf{s}_1^{x_n}(0))|
\end{aligned}$$

since $f(y,\vec{x}) \leq j(y,\vec{x})$ and $j(y,\vec{x}) = |\hat{j}(\mathsf{s}_1^y(0),\mathsf{s}_1^{x_1}(0),\ldots,\mathsf{s}_1^{x_n}(0))|$ by induction hypothesis, we are done.

Point 2 follows by point 1, Proposition 63 and (21).

□

Completeness for NB relies on a standard technique (see [BC92]), which has been adapted to the case of **FELEMENTARY** by [WW99].

Lemma 66. *For any $f(\vec{x}) \in \mathbf{FELEMENTARY}_{0,1}$ there are a function $f^*(x;\vec{x}) \in \mathsf{NB}$ and a monotone function $e_f \in \mathbf{FELEMENTARY}_{0,1}$ such that for all integers $\vec{x}$ and all $w \geq e_f(\vec{x})$ we have $f^*(w;\vec{x}) = f(\vec{x})$.*

Proof. The proof is by induction on the definition of f. If f is the zero, successor or projection function then $f^* \in \mathsf{NB}$. In this case we choose $e_f = 0$. The function ε has a definition by one application of bounded recursion on notation, proceeding from the successors and the projection functions, where each recursion is bounded by ε. Since the treatment of bounded recursion does not make use of the induction hypothesis for the bounding function, we can use this method to get functions $\varepsilon^* \in \mathsf{NB}$ and $e_\varepsilon \in \mathbf{FELEMENTARY}_{0,1}$ with the required properties. If $f(\vec{x}) = h(\vec{x},g(\vec{x}))$ then we set $f^*(w;\vec{x}) = h^*(w;\vec{x},g^*(w;\vec{x}))$, which is in NB. Since the function g^* is clearly bounded by a monotone function $b \in \mathbf{FELEMENTARY}_{0,1}$, we set $e_f(\vec{x}) = e_h(\vec{x},b(\vec{x})) + e_g(\vec{x})$, which is monotone. By applying the induction hypothesis, if $w \geq e_f(\vec{x})$ then:

$$f^*(w;\vec{x}) = h^*(w;\vec{x},g^*(w;\vec{x})) = h^*(w;\vec{x},g(\vec{x})) = h(\vec{x},g(\vec{x}))$$

Let us finally suppose that $f(x,\vec{y})$ is defined by bounded recursion on notation from $g(\vec{y})$, $h_i(x,\vec{y},f(x,\vec{y}))$ and $j(x,\vec{y})$. By applying the induction hypothesis we set:

$$\begin{aligned}\hat{f}(0,w;x,\vec{y}) &= g^*(w;\vec{y})\\ \hat{f}(\mathsf{s}_i(x),w;x,\vec{y}) &= \mathsf{cond}(;W(\mathsf{s}_i(v),w;x),g^*(w;\vec{y}),\\ &\qquad h_i^*(w;W(v,w;x),\vec{y},f^*(v,w;x,\vec{y})))\\ f^*(w;x,\vec{y}) &= \hat{f}(w,w;x,\vec{y})\end{aligned}$$

where $W(v,w;x) = \dot{-}(\dot{-}(v;w);x)$ and $\dot{-}(x;y)$ is the truncated subtraction, which is in B, and hence in NB. We can easily show that $f^*(w;x,\vec{y}) \in \mathsf{NB}$. We define $e_f(x,\vec{y}) = e_g(\vec{y}) + \sum_i e_{h_i}(x,\vec{y},j(x,\vec{y}))$, where j is the bounding function. Assuming j to be monotone, e_f is monotone too. We now show by induction on u that, whenever $w \geq e_f(x,\vec{y})$ and $w - x \leq u \leq w$:

$$\hat{f}(u,w;x,\vec{y}) = f(x-(w-u),\vec{y}) \tag{22}$$

If $u = w - x$ then we have two cases:

- if $u = 0$ then $\hat{f}(0,w;x,\vec{y}) = g^*(w;\vec{y})$;
- if $u = \mathsf{s}_i(v)$ then, since $W(\mathsf{s}_i(v),w;x) = 0$, we have $\hat{f}(\mathsf{s}_i(v),w;x,\vec{y}) = g^*(w;\vec{y})$.

Hence, in any case:

$$\hat{f}(u,w;x,\vec{y}) = g^*(w;\vec{y}) = g(\vec{y}) = f(0;\vec{y}) = f(x-(w-u),\vec{y})$$

Let us now suppose that $w - x < u \leq w$. This means that $u = \mathsf{s}_i(v)$ and $W(\mathsf{s}_i(v),w;x) > 0$. Moreover, by monotonicity of e_f and definition of j:

$$\begin{aligned}w &\geq e_f(x,\vec{y}) \geq e_f(x-(w-\mathsf{s}_i(v)),\vec{y}) \geq e_f(x-(w-v),\vec{y})\\ &\geq e_{h_i}(x-(w-v),\vec{y},j(x-(w-v),\vec{y}))\\ &\geq e_{h_i}(x-(w-v),\vec{y},f(x-(w-v),\vec{y}))\end{aligned}$$

By applying the induction hypothesis:

$$\begin{aligned}\hat{f}(\mathsf{s}_i(v),w;x,\vec{y}) &= h_i^*(w;W(v,w;x),\vec{y},\hat{f}(v,w;x,\vec{y}))\\ &= h_i^*(w;W(v,w;x),\vec{y},f(x-(w-v),\vec{y}))\\ &= h_i^*(w;x-(w-v),\vec{y},f(x-(w-v),\vec{y}))\\ &= h_i(x-(w-v),\vec{y},f(x-(w-v),\vec{y}))\\ &= f(\mathsf{s}_i((x-(w-v))),\vec{y}) = f((x-(w-\mathsf{s}_i(v))),\vec{y})\end{aligned}$$

Now, by (22), for all $w \geq e_f(x,\vec{y})$ we have:

$$f^*(w;x,\vec{y}) = \hat{f}(w,w;x,\vec{y}) = f(x,\vec{y})$$

and this concludes the proof. □

Proof of Theorem 47. First, given the function $\mathsf{ex}(x;y)$ in (4), we construct the function $\mathsf{ex}^m(x;)$ by induction on m:

$$\begin{aligned}\mathsf{ex}^1(x;) &= \mathsf{ex}(x;1)\\ \mathsf{ex}^{m+1}(x;) &= \mathsf{ex}^m(\mathsf{ex}^1(x;);)\end{aligned}$$

Hence $\varepsilon^m(x) = \mathsf{ex}^m(x;)$, for all $m \geq 1$. Now, let $f(\vec{x}) \in \mathbf{FELEMENTARY}$. By Proposition 65.1, $f(\vec{x}) \in \mathbf{FELEMENTARY}_{0,1}$. By Lemma 66, there exist

$f^*(w;\vec{x}) \in \mathsf{NB}$ and a monotone function $e_f \in \mathbf{FELEMENTARY}_{0,1}$ such that, for all $w, \vec{x}$ with $w \geq e_f(\vec{x})$, it holds that $f^*(w;\vec{x}) = f(\vec{x})$. By Proposition 65.2 there exists $m \geq 1$ such that:

$$e_f(\vec{x}) \leq \mathsf{ex}^m(\max_{\sharp\vec{x}}(\vec{x};);)$$

where $\max_{\sharp\vec{x}}(\vec{x};)$ is the k-ary maximum function, which is in B by Theorem 2, and hence in NB. Therefore:

$$f(\vec{x};) = f^*(\mathsf{ex}^m(\max_{\sharp\vec{x}}(\vec{x};););\vec{x}) \in \mathsf{NB}$$

□

Appendix D. Proofs of Section 7

Proofs of Proposition 55. By definition of cycle nf, each path from $R_{\mathcal{D}}(\nu)$ to ν in $\langle \mathcal{D}, R_{\mathcal{D}} \rangle$ is contained in a branch of $\mathcal{D}$ such that each rule instance in the former appears infinitely many times in the latter. Hence:

(i) if $\mathcal{D}$ is progressing, the path contains the conclusion of an instance of $\mathsf{cond}_{\Box N}$;
(ii) if $\mathcal{D}$ is safe, the path cannot contain the conclusion of a $\mathsf{cut}_{\Box N}$ rule;
(iii) if $\mathcal{D}$ is left-leaning, the path cannot contain the rightmost premise of a cut_N rule.

This shows point 1. Let us consider point 2. By point (ii), if $\mathcal{D}$ is safe then, going from a node μ of the path to each of its children μ', the number of modal formulas in the context of the corresponding sequents cannot increase. Moreover, the only cases where this number strictly decreases is when μ is the conclusion of $\Box_l$, $\mathsf{w}_{\Box N}$, or when μ' is the leftmost premise of $\mathsf{cond}_{\Box N}$. Since $R_{\mathcal{D}}(\nu)$ and ν must be labelled with the same sequent, all such cases are impossible. As for point 3 we notice that, by point (iii) and the above reasoning, if $\mathcal{D}$ is safe and left-leaning then, going from a node μ of the path to each of its children μ', the number of non-modal formulas in the context of the corresponding sequents cannot increase. Moreover, the only cases where this number strictly decreases is when μ is the conclusion of w_N, or when μ' is the leftmost premise of cond_N. Since $R_{\mathcal{D}}(\nu)$ and ν must be labelled with the same sequent, all such cases are impossible. □

Proof of Lemma 57. Points 1 and 2 are proven by simultaneous induction on the longest distance of ν_0 from a leaf of $T_{\mathcal{D}}$. Notice that in the following situations only point 1 applies:

- ν is the conclusion of an instance of id or 0;
- ν is the conclusion of an instance of $\mathsf{w}_{\Box}$, $\Box_l$ and $\mathsf{cut}_{\Box N}$;
- ν is the conclusion of an instance of w_N and $\mathcal{D}$ is a CB-coderivation.

In particular, the last two cases hold by Proposition 55.2-3, as it must be that $O_{\nu'} = \varnothing$ for any premise ν' of ν_0. Let us discuss the case where ν_0 is the conclusion of a $\mathsf{cut}_{\Box}$ rule with premises ν_1 and ν_2. By induction on point 1 we have $f_{\mathcal{D}_{\nu_1}}(\vec{x};\vec{y}), f_{\mathcal{D}_{\nu_2}}(\vec{x},x;\vec{y}) \in \mathsf{NB}^{\subset}$. Since the conclusion of $\mathcal{D}_{\nu_1}$ has modal succedent, by Proposition 18 there must be a coderivation $\mathcal{D}^*$ such that $f_{\mathcal{D}^*}(\vec{x};) = f_{\mathcal{D}_{\nu_1}}(\vec{x};\vec{y}) \in \mathsf{NB}^{\subset}$. Moreover, by Proposition 31, if $\mathcal{D}_{\nu_1}$ is a CB-coderivation then $\mathcal{D}^*$ is. Hence, we define $f_{\mathcal{D}_{\nu_0}}(\vec{x};\vec{y}) = f_{\mathcal{D}_{\nu_2}}(\vec{x}, f_{\mathcal{D}^*}(\vec{x};);\vec{y}) \in \mathsf{NB}^{\subset}$. If moreover $\mathcal{D}$ is a CB-coderivation then, by applying the induction hypothesis, we obtain $f_{\mathcal{D}_{\nu_0}} \in \mathsf{B}^{\subset}$.

Let us now consider point 2. If ν_0 is the conclusion of a bud then $O_{\nu_0} = \{\nu_0\}$, $C_{\nu_0} = \varnothing$, and all points hold trivially. The cases where ν_0 is an instance of w_N, e_N, $\mathsf{e}_\Box$, $\Box_r$, s_0 or s_1 are straightforward. Suppose that ν_0 is the conclusion of a $\mathsf{cond}_\Box$ step with premises ν', ν_1, and ν_2, and let us assume $O_{\nu_1} \neq \varnothing$, $O_{\nu_2} \neq \varnothing$. By Proposition 55.2 we have $O_{\nu'} = \varnothing$, so that $f_{\mathcal{D}_{\nu'}} \in \mathsf{NB}^\subset$ by induction hypothesis on point 1. By definition, $O_{\nu_0} = O_{\nu_1} \cup O_{\nu_2}$ and $C_{\nu_0} = C_{\nu_1} \cup C_{\nu_2}$. Then, we set:

$$f_{\mathcal{D}_{\nu_0}}(x, \vec{x}; \vec{y}) = \mathsf{cond}(; x, f_{\mathcal{D}_{\nu'}}(\vec{x}; \vec{y}), f_{\mathcal{D}_{\nu_1}}(\mathsf{p}(x;), \vec{x}; \vec{y}), f_{\mathcal{D}_{\nu_2}}(\mathsf{p}(x;), \vec{x}; \vec{y}))$$

where $\mathsf{p}(x;)$ can be defined from $\mathsf{p}(; x)$ and projections. By induction hypothesis on ν_i:

$$\begin{aligned} f_{\mathcal{D}_{\nu_i}}(\mathsf{p}(x;), \vec{x}; \vec{y}) &= h_{\nu_i}((\lambda u, \vec{u} \subseteq \mathsf{p}(x;), \vec{x}, \lambda \vec{v}. f_{\mathcal{D}_\mu}(u, \vec{u}; \vec{v}))_{\mu \in C_{\nu_i} \cup O_{\nu_i}})(\mathsf{p}(x;), \vec{x}; \vec{y}) \\ &= h_{\nu_i}((\lambda u, \vec{u} \subset x, \vec{x}, \lambda \vec{v}. f_{\mathcal{D}_\mu}(u, \vec{u}; \vec{v}))_{\mu \in C_{\nu_i} \cup O_{\nu_i}})(\mathsf{p}(x;), \vec{x}; \vec{y}) \end{aligned}$$

hence $f_{\mathcal{D}_{\nu_0}}(x, \vec{x}; \vec{y}) = h_{\nu_0}((\lambda u, \vec{u} \subset x, \vec{x}, \lambda \vec{v}. f_{\mathcal{D}_\mu}(u, \vec{u}; \vec{v}))_{\mu \in C_{\nu_0} \cup O_{\nu_0}})(x, \vec{x}; \vec{y})$, for some h_{ν_0}. By applying the induction hypothesis, we have $h_{\nu_0} \in \mathsf{NB}^\subset((f_{\mathcal{D}_\mu})_{\mu \in O_{\nu_0}}, (a_\mu)_{\mu \in C_{\nu_0}})$ and $f_{\mathcal{D}_{\nu_0}} \in \mathsf{NB}^\subset((f_{\mathcal{D}_\mu})_{\mu \in O_{\nu_0}})$. This shows point 2a. Point 2b is trivial, and point 2c holds by applying the induction hypothesis.

Let us now consider the case where ν_0 is an instance of cond_N, assuming $O_{\nu_1} \neq \varnothing$ and $O_{\nu_2} \neq \varnothing$. The only interesting case is when $\mathcal{D}$ is a CB-coderivation. By Proposition 55.3 we have $O_{\nu'} = \varnothing$, so that $f_{\mathcal{D}_{\nu'}} \in \mathsf{B}^\subset$ by induction hypothesis on point 1. By definition, $O_{\nu_0} = O_{\nu_1} \cup O_{\nu_2}$ and $C_{\nu_0} = C_{\nu_1} \cup C_{\nu_2}$. Then, we set:

$$f_{\mathcal{D}_{\nu_0}}(\vec{x}; y, \vec{y}) = \mathsf{cond}(; y, f_{\mathcal{D}_{\nu'}}(\vec{x}; \vec{y}), f_{\mathcal{D}_{\nu_1}}(\vec{x}; \mathsf{p}(; y), \vec{y}), f_{\mathcal{D}_{\nu_2}}(\vec{x}; \mathsf{p}(; y), \vec{y}))$$

By induction hypothesis on ν_i:

$$\begin{aligned} &f_{\mathcal{D}_{\nu_i}}(\vec{x}; \mathsf{p}(; y), \vec{y}) = \\ &= h_{\nu_i}((\lambda \vec{u} \subseteq x, \vec{x}, \lambda v, \vec{v} \subseteq \mathsf{p}(; y), \vec{y}. f_{\mathcal{D}_\mu}(\vec{u}; v, \vec{v}))_{\mu \in C_{\nu_i} \cup O_{\nu_i}})(\vec{x}; \mathsf{p}(; y), \vec{y}) \\ &= h_{\nu_i}((\lambda \vec{u} \subset \vec{x}, \lambda v, \vec{v} \subset y, \vec{y}. f_{\mathcal{D}_\mu}(\vec{u}; v, \vec{v}))_{\mu \in C_{\nu_i} \cup O_{\nu_i}})(\vec{x}; \mathsf{p}(; y), \vec{y}) \end{aligned}$$

hence $f_{\mathcal{D}_{\nu_0}}(\vec{x}; y, \vec{y}) = h_{\nu_0}((\lambda \vec{u} \subseteq \vec{x}, \lambda v, \vec{v} \subset y, \vec{y}. f_{\mathcal{D}_\mu}(\vec{u}; v, \vec{v}))_{\mu \in C_{\nu_0} \cup O_{\nu_0}})(\vec{x}; y, \vec{y})$, for some h_{ν_0}. This shows point 2c. Point 2a and 2b are given by the induction hypothesis.

Let us now consider the case where ν_0 is the conclusion of an instance of dis with premise ν', where X is the set of nodes labelling the rule. We have $O_{\nu_0} = O_{\nu'} \setminus X$ and $C_{\nu_0} = C_{\nu'} \cup \{\nu_0\}$. We want to find $(h_\nu)_{\nu \in C_\nu \cup \{\nu_0\}}$ defining the equations for $(f_{\mathcal{D}_\nu})_{\nu \in C_\nu \cup \{\nu_0\}}$ in such a way that points 2a-2c hold. We shall start by defining h_{ν_0}. First, note that, by definition of cycle nf, $f_{\mathcal{D}_{\nu_0}}(\vec{x}; \vec{y}) = f_{\mathcal{D}_{\nu'}}(\vec{x}; \vec{y}) = f_{\mathcal{D}_\mu}(\vec{x}; \vec{y})$ for all $\mu \in X$. By induction hypothesis on ν' there exists a family $(g_\nu)_{\nu \in C_{\nu'} \cup \{\nu'\}}$ such that:

$$f_{\mathcal{D}_{\nu'}}(\vec{x}; \vec{y}) = g_{\nu'}((\lambda \vec{u} \subseteq \vec{x}, \lambda \vec{v}. f_{\mathcal{D}_\mu}(\vec{u}; \vec{v}))_{\mu \in C_{\nu'} \cup O_{\nu'}})(\vec{x}; \vec{y}) \tag{23}$$

and, moreover, for all $\nu \in C_{\nu'}$:

$$f_{\mathcal{D}_\nu}(\vec{x}; \vec{y}) = g_\nu((\lambda \vec{u} \subseteq \vec{x}, \lambda \vec{v}. f_{\mathcal{D}_\mu}(\vec{u}; \vec{v}))_{\mu \in C_{\nu'} \cup O_{\nu'}})(\vec{x}; \vec{y}) \tag{24}$$

Since $O_{\nu'} = O_{\nu_0} \cup X$ and the path from ν' to any $\mu \in X$ must cross an instance of $\mathsf{cond}_\Box$ by Proposition 55.1, the induction hypothesis on ν' (point 2b) allows us to

rewrite (23) as follows:

$$\begin{aligned} f_{\mathcal{D}_{\nu'}}(\vec{x};\vec{y}) = g_{\nu'}((&\lambda\vec{u}\subseteq\vec{x},\lambda\vec{v}.f_{\mathcal{D}_\nu}(\vec{u};\vec{v}))_{\nu\in C_{\nu'}},\\ &(\lambda\vec{u}\subseteq\vec{x},\lambda\vec{v}.f_{\mathcal{D}_\mu}(\vec{u};\vec{v}))_{\mu\in O_{\nu_0}},\\ &(\lambda\vec{u}\subset\vec{x},\lambda\vec{v}.f_{\mathcal{D}_\mu}(\vec{u};\vec{v}))_{\mu\in X})(\vec{x};\vec{y}) \end{aligned} \tag{25}$$

On the other hand, for all $\nu\in\mathcal{C}_{\nu'}$, for all $\vec{u}\subseteq\vec{x}$ and $\vec{v}$, the equation in (24) can be rewritten as:

$$\begin{aligned} f_{\mathcal{D}_\nu}(\vec{u};\vec{v}) = g_\nu((&\lambda\vec{w}\subset\vec{u},\lambda\vec{v'}.f_{\mathcal{D}_\mu}(\vec{u};\vec{v'}))_{\mu\in C_{\nu'}},\\ &(\lambda\vec{w}\subseteq\vec{u},\lambda\vec{v'}.f_{\mathcal{D}_\mu}(\vec{w};\vec{v'}))_{\mu\in O_{\nu_0}},\\ &(\lambda\vec{w}\subseteq\vec{u},\lambda\vec{v'}.f_{\mathcal{D}_\mu}(\vec{w};\vec{v'}))_{\mu\in X})(\vec{u};\vec{v}) \end{aligned} \tag{26}$$

and so, for all $\nu\in C_{\nu'}$:

$$\begin{aligned} \lambda\vec{u}\subseteq\vec{x},\lambda v.f_{\mathcal{D}_\nu}(\vec{u};\vec{v}) = \lambda\vec{u}\subseteq\vec{x},\lambda v.g_\nu((&\lambda\vec{w}\subset\vec{x},\lambda\vec{v'}.f_{\mathcal{D}_\mu}(\vec{u};\vec{v'}))_{\mu\in C_{\nu'}},\\ &(\lambda\vec{w}\subseteq\vec{x},\lambda\vec{v'}.f_{\mathcal{D}_\mu}(\vec{w};\vec{v'}))_{\mu\in O_{\nu_0}},\\ &(\lambda\vec{w}\subseteq\vec{x},\lambda\vec{v'}.f_{\mathcal{D}_\mu}(\vec{w};\vec{v'}))_{\mu\in X})(\vec{u};\vec{v}) \end{aligned} \tag{27}$$

Now, since the paths from ν' to any $\mu\in X$ in $\mathcal{D}$ must contain an instance of $\mathsf{cond}_\square$, for all $\nu\in C_{\nu'}$ and all $\mu\in X$, we have that either the path from ν' to ν contains an instance of $\mathsf{cond}_\square$ or the path from ν to μ does. By applying the induction hypothesis on ν' (point 2b), given $\nu\in C_{\nu'}$ and $\mu\in X$, either $\lambda\vec{u}\subseteq\vec{x},\lambda\vec{v}.f_{\mathcal{D}_\nu}(\vec{u};\vec{v})$ in (25) is such that $\vec{u}\subset\vec{x}$, or $\lambda\vec{w}\subseteq\vec{u},\lambda\vec{v'}.f_{\mathcal{D}_\mu}(\vec{w};\vec{v'})$ in (26) is such that $\vec{w}\subset\vec{u}$. This means that, for any $\mu\in X$, $\lambda\vec{w}\subseteq\vec{x},\lambda\vec{v'}.f_{\mathcal{D}_\mu}(\vec{w};\vec{v'})$ in (27) is such that $\vec{w}\subset\vec{x}$. For each $\nu\in C_{\nu'}$, by rewriting $\lambda\vec{u}\subseteq\vec{x},\lambda\vec{v}.f_{\mathcal{D}_\nu}(\vec{u};\vec{v})$ in (25) according to the equation in (27) we obtain:

$$f_{\mathcal{D}_{\nu_0}}(\vec{x};\vec{y}) = t_{\nu_0}((\lambda\vec{u}\subset\vec{x},\lambda\vec{v}.f_{\mathcal{D}_\mu}(\vec{u};\vec{v}))_{\mu\in C_{\nu'}\cup X},(\lambda\vec{u}\subseteq\vec{x},\lambda\vec{v}.f_{\mathcal{D}_\mu}(\vec{u};\vec{v}))_{\mu\in O_{\nu_0}})(\vec{x};\vec{y})$$

for some t_{ν_0}. Since $f_{\mathcal{D}_\mu}=f_{\mathcal{D}_{\nu_0}}$ for all $\mu\in X$, and since $C_{\nu_0}=C_{\nu'}\cup\{\nu_0\}$, by setting $h_{\nu_0}:=t_{\nu_0}$ the above equation gives us the following:

$$f_{\mathcal{D}_{\nu_0}}(\vec{x};\vec{y}) = h_{\nu_0}((\lambda\vec{u}\subseteq\vec{x},\lambda\vec{v}.f_{\mathcal{D}_\mu}(\vec{u};\vec{v}))_{\mu\in C_{\nu_0}\cup O_{\nu_0}})(\vec{x};\vec{y}) \tag{28}$$

which satisfies point 2b. From (28) we are able to find the functions $(h_\nu)_{\nu\in C_\nu}$ defining the equations for $(f_{\mathcal{D}_\nu})_{\nu\in C_\nu}$. Indeed, the induction hypothesis on ν' gives us (24) for any $\nu\in C_{\nu'}$. We rewrite in each such equation any $\lambda\vec{u}\subseteq\vec{x},\lambda\vec{v}.f_{\mathcal{D}_\mu}(\vec{u};\vec{v})$ such that $\mu\in X$ according to equation (28), as $f_{\mathcal{D}_\mu}=f_{\mathcal{D}_{\nu_0}}$ for any $\mu\in X$. We obtain the following equation for any $\nu\in C_{\nu'}$:

$$f_{\mathcal{D}_\nu}(\vec{x};\vec{y}) = t_\nu((\lambda\vec{u}\subset\vec{x},\lambda\vec{v}.f_{\mathcal{D}_\mu}(\vec{u};\vec{v}))_{\mu\in C_{\nu'}\cup\{\nu_0\}},(\lambda\vec{u}\subseteq\vec{x},\lambda\vec{v}.f_{\mathcal{D}_\mu}(\vec{u};\vec{v}))_{\mu\in O_{\nu_0}})(\vec{x};\vec{y})$$

for some t_ν. Since $C_{\nu_0}=C_{\nu'}\cup\{\nu_0\}$ and the above equation satisfies point 2b, we set $h_\nu:=t_\nu$ and we obtain, for all $\nu\in C_{\nu_0}$:

$$f_{\mathcal{D}_\nu}(\vec{x};\vec{y}) = h_\nu((\lambda\vec{u}\subseteq\vec{x},\lambda\vec{v}.f_{\mathcal{D}_\mu}(\vec{u};\vec{v}))_{\mu\in C_{\nu_0}\cup O_{\nu_0}})(\vec{x};\vec{y}) \tag{29}$$

It remains to show that (28) and (29) satisfy points 2a and 2c. Concerning point 2a, on the one hand for all $\nu\in C_{\nu_0}$ we have $h_\nu\in\mathsf{NB}^\subset((f_{\mathcal{D}_\mu})_{\mu\in O_{\nu_0}},(a_\mu)_{\mu\in C_{\nu_0}})$, with $(a_\mu)_{\mu\in C_{\nu_0}}$ oracle functions. On the other hand, by applying the induction hypothesis, we have $f_{\mathcal{D}_{\nu_0}}=f_{\mathcal{D}_{\nu'}}\in\mathsf{NB}^\subset((f_{\mathcal{D}_\mu})_{\mu\in O_{\nu'}})$ and $f_{\mathcal{D}_\nu}\in\mathsf{NB}^\subset((f_{\mathcal{D}_\mu})_{\mu\in O_{\nu'}})$, for all $\nu\in C_{\nu_0}$. Since $O_{\nu'}=O_{\nu_0}\cup X$ and $f_{\mathcal{D}_{\nu_0}}=f_{\mathcal{D}_\mu}$ for all $\mu\in X$, we have both $f_{\mathcal{D}_{\nu_0}}\in\mathsf{NB}^\subset((f_{\mathcal{D}_\mu})_{\mu\in O_{\nu_0}})$ and $f_{\mathcal{D}_\nu}\in\mathsf{NB}^\subset((f_{\mathcal{D}_\mu})_{\mu\in O_{\nu_0}},f_{\mathcal{D}_{\nu_0}})$ for all $\nu\in C_{\nu_0}$, and

hence $f_{\mathcal{D}_\nu} \in \mathsf{NB}^{\subset}((f_{\mathcal{D}_\mu})_{\mu \in O_{\nu_0}})$, for all $\nu \in C_{\nu_0}$. Point 2c follows by applying the induction hypothesis, as the construction does not affect the safe arguments.

Last, suppose that ν_0 is the conclusion of an instance of cut_N with premises ν_1 and ν_2. We shall only consider the case where $O_{\nu_i} \neq \varnothing$ for $i = 1, 2$. Then, $O_{\nu_0} = O_{\nu_1} \cup O_{\nu_2}$ and $C_{\nu_0} = C_{\nu_1} \cup C_{\nu_2}$. We have:

$$f_{\mathcal{D}_{\nu_0}}(\vec{x}; \vec{y}) = f_{\mathcal{D}_{\nu_2}}(\vec{x}; f_{\mathcal{D}_{\nu_1}}(\vec{x}; \vec{y}), \vec{y})$$

By induction hypothesis on ν_1 and ν_2:

$$f_{\mathcal{D}_{\nu_1}}(\vec{x}; \vec{y}) = h_{\nu_1}((\lambda \vec{u} \subseteq \vec{x}, \lambda \vec{v}. f_{\mathcal{D}_\mu}(\vec{u}; \vec{v}))_{\mu \in C_{\nu_1} \cup O_{\nu_1}})(\vec{x}; \vec{y})$$
$$f_{\mathcal{D}_{\nu_2}}(\vec{x}; y, \vec{y}) = h_{\nu_2}((\lambda \vec{u} \subseteq \vec{x}, \lambda v, \vec{v}. f_{\mathcal{D}_\mu}(\vec{u}; v, \vec{v}))_{\mu \in C_{\nu_2} \cup O_{\nu_2}})(\vec{x}; y, \vec{y})$$

So that:

$$f_{\mathcal{D}_{\nu_0}}(\vec{x}; \vec{y}) = h_{\nu_0}((\lambda \vec{u} \subseteq \vec{x}, \lambda \vec{v}. f_{\mathcal{D}_\mu}(\vec{u}; \vec{v}))_{\mu \in C_{\nu_0} \cup O_{\nu_0}})(\vec{x}; \vec{y})$$

Points 2a and 2b hold by applying the induction hypothesis. Concerning point 2c, notice that if $\mathcal{D}$ is a CB-coderivation then $O_{\nu_2} = \varnothing$ by Proposition 55.3. By applying the induction hypothesis on ν_1 and ν_2, we have $f_{\mathcal{D}_{\nu_2}}(\vec{x}; y, \vec{y}) \in \mathsf{B}^{\subset}$ and $f_{\mathcal{D}_{\nu_1}}(\vec{x}; \vec{y}) = h_{\nu_1}((\lambda \vec{u} \subseteq \vec{x}, \lambda \vec{v} \subseteq \vec{y}. f_{\mathcal{D}_\mu}(\vec{u}; \vec{v}))_{\mu \in C_{\nu_1} \cup O_{\nu_1}})(\vec{x}; \vec{y})$, so that:

$$f_{\mathcal{D}_{\nu_0}}(\vec{x}; \vec{y}) = h_{\nu_0}((\lambda \vec{u} \subseteq \vec{x}, \lambda \vec{v} \subseteq \vec{y}. f_{\mathcal{D}_\mu}(\vec{u}; \vec{v}))_{\mu \in C_{\nu_0} \cup O_{\nu_0}})(\vec{x}; \vec{y})$$

for some h_{ν_0}. □

Appendix E. Further examples

Example 67 (Predecessor)**.** The unary predecessor function can be defined by the following coderivation $\mathcal{P}$

$$\dfrac{\dfrac{}{0 \Rightarrow N}\,0 \qquad \dfrac{\dfrac{\dfrac{\vdots}{\Box N \Rightarrow N}\,\mathsf{cond}_{\Box N}\bullet \qquad \dfrac{\dfrac{}{N \Rightarrow N}\,\mathsf{id}}{N \Rightarrow N}\,\mathsf{s}_1}{\Box N \Rightarrow \Box N}\,\Box_r \qquad \dfrac{\dfrac{}{N \Rightarrow N}\,\mathsf{id}}{\Box N \Rightarrow N}\,\Box_l}{\Box N \Rightarrow N}\,\mathsf{cut}_{\Box N} \qquad \dfrac{\dfrac{}{N \Rightarrow N}\,\mathsf{id}}{\Box N \Rightarrow N}\,\mathsf{s}_0}{\Box N \Rightarrow N}\,\Box_l}{\Box N \Rightarrow N}\,\mathsf{cond}_{\Box N}\bullet$$

The equational program associated with the above coderivation can be written as follows:

$$\begin{array}{rcll} f_{\mathcal{P}_\epsilon}(0;) & = & 0 & \\ f_{\mathcal{P}_\epsilon}(\mathsf{s}_0 x; y) & = & \mathsf{s}_1 f_{\mathcal{P}_\epsilon}(x;) & x \neq 0 \\ f_{\mathcal{P}_\epsilon}(\mathsf{s}_1 x; y) & = & \mathsf{s}_0 x & \end{array}$$

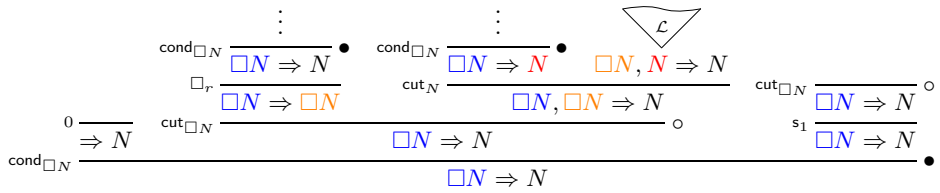
Example 68 (Sum of lengths)**.** The function returning the sum of the binary lengths of two natural numbers can be defined by the following coderivation $\mathcal{L}$:

$$\dfrac{\dfrac{}{N \Rightarrow N}\,\mathsf{id} \qquad \dfrac{\dfrac{\dfrac{\vdots}{\Box N, N \Rightarrow N}\,\mathsf{cond}_{\Box N}\bullet \qquad \dfrac{\dfrac{}{N \Rightarrow N}\,\mathsf{id}}{N \Rightarrow N}\,\mathsf{s}_1}{\Box N, N \Rightarrow N}\,\mathsf{cut}_N}{\Box N, N \Rightarrow N}\,\mathsf{cond}_{\Box N}\bullet$$

where we identified the sub-coderivations corresponding to the second and the third premises of the conditional rule. We obtain the following equational program:

$$\begin{array}{rcll} f_{\mathcal{L}_\epsilon}(0;y) & = & y & \\ f_{\mathcal{L}_\epsilon}(\mathsf{s}_0 x;y) & = & \mathsf{s}_1 f_{\mathcal{L}_\epsilon}(x;y) & x \neq 0 \\ f_{\mathcal{L}_\epsilon}(\mathsf{s}_1 x;y) & = & \mathsf{s}_1 f_{\mathcal{L}_\epsilon}(x;y) & \end{array}$$

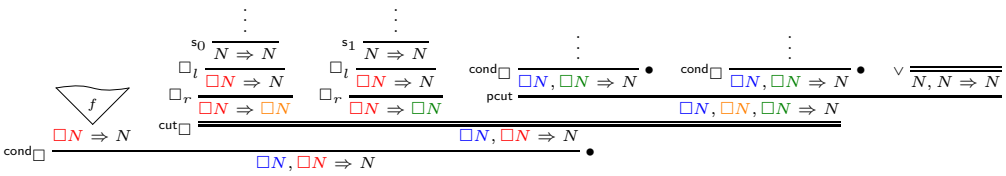
Example 69 (Binary to unary)**.** The function converting n into $\mathsf{s}_1.\overset{n}{.}.\mathsf{s}_1 0$ can be defined by the following coderivation $\mathcal{N}$:



where $\mathcal{L}$ is the coderivation in Example 68, and $\circ$ indicates roots of identical sub-coderivations. The equational program associated to the above coderivation is the following:

$$\begin{array}{rcll} f_{\mathcal{N}_\epsilon}(0;) & = & 0 & \\ f_{\mathcal{N}_\epsilon}(\mathsf{s}_0 x;) & = & f_{\mathcal{L}}(f_{\mathcal{N}_\epsilon}(x;); f_{\mathcal{N}_\epsilon}(x;)) & x \neq 0 \\ f_{\mathcal{N}_\epsilon}(\mathsf{s}_1 x;) & = & \mathsf{s}_1(f_{\mathcal{L}}(f_{\mathcal{N}_\epsilon}(x;); f_{\mathcal{N}_\epsilon}(x;))) & \end{array}$$

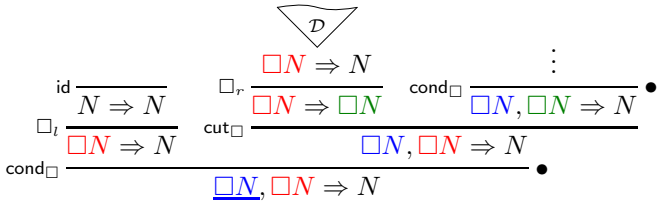
Example 70 (Polynomially bounded search)**.** Let $f(x;)$ be a 0-1-valued function (say, in B), and let $\vee(;x,y)$ be the function $\mathsf{cond}(;x,y,y,1)$. Then, polynomially bounded search can be defined by the coderivation $\mathcal{B}$ using parallel cut (see Section 8):



where we identified the sub-coderivations corresponding to the second and the third premise of the conditional step, and double line means multiple application of the same rule. The equational program can be rewritten as follows:

$$\begin{array}{rcll} f_{\mathcal{B}_\epsilon}(0,y;) & = & f(y;) & \\ f_{\mathcal{B}_\epsilon}(\mathsf{s}_0 x,y;) & = & \vee(; f_{\mathcal{B}_\epsilon}(x,\mathsf{s}_0 y;), f_{\mathcal{B}_\epsilon}(x,\mathsf{s}_1 y;)) & x \neq 0 \\ f_{\mathcal{B}_\epsilon}(\mathsf{s}_1 x,y;) & = & \vee(; f_{\mathcal{B}_\epsilon}(x,\mathsf{s}_0 y;), f_{\mathcal{B}_\epsilon}(x,\mathsf{s}_1 y;)) & \end{array}$$

Example 71 (Counterexample to a weaker safety condition)**.** Consider the following coderivation $\mathcal{E}'$ of CNB:



where $f_{\mathcal{D}}(x;) = \mathsf{ex}(x;1)$ (see Example 27) and we identified the sub-coderivations corresponding to the second and the third premise of the conditional step. The associated equational program can be rewritten as follows:

$$\begin{array}{rcll} f_{\mathcal{E}'_\epsilon}(0,y;) & = & y & \\ f_{\mathcal{E}'_\epsilon}(\mathsf{s}_0 x,y;) & = & f_{\mathcal{E}'_\epsilon}(x, f_{\mathcal{D}}(y;);) & x \neq 0 \\ f_{\mathcal{E}'_\epsilon}(\mathsf{s}_1 x,y;) & = & f_{\mathcal{E}'_\epsilon}(x, f_{\mathcal{D}}(y;);) & \end{array}$$

It is not hard to see that the above program is not bounded by any elementary function. This example and Example 19 illustrate that allowing loops to cross the rightmost (resp. leftmost) premise of a $\mathsf{cut}_\square$ produce non-elementary computable functions. Therefore, the safety condition cannot be relaxed.