Cyclic Implicit Complexity

GIANLUCA CURZI, University of Gothenburg, Sweden and University of Birmingham, UK
ANUPAM DAS, University of Birmingham, UK

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, several new 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 (ICC). Namely we introduce a circular proof system based on Bellantoni and Cook’s famous safe-normal function algebra, and we identify proof theoretical constraints, inspired by ICC, to characterise the polynomial-time and elementary computable functions. Along the way we introduce new recursion theoretic implicit characterisations of these classes that may be of interest in their own right.

CCS Concepts: • Theory of computation → Complexity theory and logic; Proof theory.

Additional Key Words and Phrases: Cyclic proofs, implicit complexity, function algebras, safe recursion, higher-order complexity

ACM Reference Format:
Gianluca Curzi and Anupam Das. 2018. Cyclic Implicit Complexity. In Woodstock ’18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY. ACM, New York, NY, USA. 45 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Formal proofs are traditionally seen as finite objects modelling logical or mathematical reasoning. Non-wellfounded proofs are a generalisation of this notion to an infinitary (but finitely branching) setting, in which consistency is maintained by a standard global condition: the ‘progressing’ criterion. Special attention is devoted to regular (or circular or cyclic) proofs, i.e. those non-wellfounded proofs having only finitely many distinct sub-proofs, and which may thus be represented by finite (possibly cyclic) directed graphs. For such proofs the progressing criterion may be effectively decided by reduction to the universality problem for Büchi automata.

Non-wellfounded proofs have been employed to reason about the modal μ-calculus and fixed-point logics [19, 30], first-order inductive definitions [10], Kleene algebra [16, 17], linear logic [3, 20], arithmetic [8, 14, 34], system T [13, 15, 25], and continuous cut-elimination [2, 21, 29]. In particular, [25] and [13, 15] investigate the computational expressivity of circular proofs, with respect to the proofs-as-programs paradigm, in the setting of higher-order primitive recursion.

However little is known about the complexity-theoretic aspects of circular proofs. Usual termination arguments for circularly typed programs are nonconstructive, proceeding by contradiction and using a non-recursive ‘totality’ oracle (cf. [13, 15, 25]). As a result, these arguments are not appropriate for delivering feasible complexity bounds (cf. [14]).

The present paper aims to bridge this gap by proposing a circular foundation for Implicit Computational Complexity (ICC), a branch of computational complexity studying machine-free characterisations of complexity classes. Our starting point is Bellantoni and Cook’s famous function algebra B characterising the polynomial time computable...
functions (FPTIME) using safe recursion [6]. The prevailing idea behind safe recursion (and its predecessor, 'ramified' recursion [26]) is to organise data into strata in a way that prevents recursive calls being substituted into recursive parameters of previously defined functions. This approach has been successfully employed to give resource-bound-free characterisations of polynomial-time [6], levels of the polynomial-time hierarchy [5], and levels of the Grzegorczyk hierarchy [35], and has been extended to higher-order settings too [22, 27].

Circular systems for implicit complexity

Construing \( B \) as a type system, we consider non-wellfounded proofs, or coderivations, generated by its recursion-free subsystem \( B^- \). The circular proof system \( \text{CNB} \) is then obtained by considering the regular and progressing coderivations of \( B^- \) which satisfy a further criterion, safety, motivated by the eponymous notion from Bellantoni and Cook’s work (cf. also ‘ramification’ in Leivant’s work [26]). On the one hand, regularity and progressiveness ensure that coderivations of \( \text{CNB} \) define total computable functions; on the other hand, the latter criterion ensures that the corresponding equational programs are ‘safe’: the recursive call of a function is never substituted into the recursive parameter of a step function.

Despite \( \text{CNB} \) having only ground types, it is able to define equational programs that nest recursive calls, a property that typically arises only in higher-order recursion (cf., e.g., [22, 27]). In fact, we show that this system defines precisely the elementary computable functions (FELEMENTARY). Let us point out that the capacity of circular proofs to simulate some higher-order behaviour reflects an emerging pattern in the literature. For instance in [13, 15] is shown that the number-theoretic functions definable by type level \( n \) proofs of a circular version of system \( T \) are exactly those definable by type level \( n + 1 \) proofs of \( T \).

In the setting of ICC, Hofmann [22] and Leivant [27] already observed that higher-order safe recursion mechanisms can be used to characterise FELEMENTARY. In particular, in [22], Hofmann presents the type system SLR (Safe Linear Recursion) as a higher-order version of \( B \) imposing a ‘linearity’ restriction on the higher-order safe recursion operator. He shows that this system defines just the polynomial-time computable functions on natural numbers (FPTIME).

Inspired by [22], we too introduce a linearity requirement for \( \text{CNB} \) that is able to control the interplay between cycles and the cut rule, called the left-leaning criterion. The resulting circular proof system is called CB, which we show defines precisely FPTIME.

Function algebras for safe nested recursion and safe recursion along well-founded relations

As well as introducing the circular systems CB and \( \text{CNB} \) just mentioned, we also develop novel function algebras for FPTIME and FELEMENTARY that allow us to prove the aforementioned complexity characterisations via a ‘sandwich’ technique, cf. Figure 1. This constitutes a novel (and more direct) approach to reducing circularity to recursion, that crucially takes advantage of safety.

We give a relativised formulation of \( B \), i.e. with oracles, that allows us to define a form of safe nested recursion. The resulting function algebra is called NB and is comparable to the type 2 fragment of Leivant’s extension of ramified recursion to finite types [27]. The algebras \( B \) and \( \text{NB} \) will serve as lower bounds for CB and \( \text{CNB} \) respectively.

The relativised formulation of function algebras also admits a robust notion of (safe) recursion along a well-founded relation. We identify a particular well-founded preorder \( \subset \) (‘permutation of prefixes’) whose corresponding safe recursor induces algebras \( B^- \) and \( \text{NB}^- \) that will serve as upper bounds for CB and \( \text{CNB} \) respectively.
Outline

This paper is structured as follows. Section 2 presents \( B \) as a proof system. In Section 3 we define non-wellfounded proofs and their semantics and present the circular proof systems \( CNB \) and \( CB \). In Section 4 we present the function algebras \( NB, B^C \) and \( NB^C \). Section 5 shows that \( B^C \) captures \( \text{FPTIME} \) (Corollary 43) and that both \( NB \) and \( NB^C \) capture \( \text{FELEMENTARY} \) (Corollary 50). These results require a delicate Bounding Lemma (Lemma 40) and an encoding of the elementary functions into \( NB \) (Theorem 49). In Section 6 we show that any function definable in \( B \) is also definable in \( CB \) (Theorem 51), and that any function definable in \( NB \) is also definable in \( CNB \) (Theorem 54). Finally, in Section 6.2, we present a translation of \( CNB \) into \( NB^C \) that maps \( CB \) coderivations into \( B^C \) functions (Lemma 55), by reducing circularity to a form of simultaneous recursion on \( C \).

The main results of this paper are summarised in Figure 1.

Comparison with conference version

This article is a full version of the conference paper [11], which appeared in the proceedings of Logic in Computer Science 2022. The current version expands upon [11] by including full proofs, as well as many further examples and remarks to aid the narrative.

2 PRELIMINARIES

Bellantoni and Cook introduced in [6] an algebra of functions based on a simple two-sorted structure. This idea was itself inspired by Leivant’s characterisations, one of the founding works in Implicit Computational Complexity (ICC) [26]. 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 \( \bar{x} \) and \( n \) safe inputs \( \bar{y} \). 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 \( \bar{x} = x_1, \ldots, x_m \) we write \(|\bar{x}|\) for the list \(|x_1|, \ldots, |x_m|\).

2.1 Bellantoni-Cook characterisation of \( \text{FPTIME} \)

We first recall Bellantoni-Cook in its original guise.

Definition 1 (Bellantoni-Cook algebra). \( B \) is defined as the smallest class of (two-sorted) functions containing,
We may write $D \Gamma, A, B,$ etc. to vary over types.

2.2 Proof theoretic presentation of Bellantoni-Cook

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 $D, E, \ldots$. We write $D : \Gamma \Rightarrow A$ if the derivation $D$ has end-sequent $\Gamma \Rightarrow A$. We may write $D = r(D_1, \ldots, D_n)$ (for $n \leq 3$) if $r$ is the last inference step of $D$ whose immediate subderivations are, respectively, $D_1, \ldots, D_n$. Unless otherwise indicated, we assume that the $r$-instance is as typeset in Figure 2.

Convention 3 (Left normal, right safe). In what follows, we assume that sequents have shape $\Box N, \ldots, \Box N, N, \ldots, N \Rightarrow A$, i.e. in the LHS all $\Box N$ occurrences are placed before all $N$ occurrences. Note that this invariant is maintained by the
We construe the system of $\text{B}$-derivations as a class of safe-normal functions by identifying each rule instance as an operation on safe-normal functions. Formally:

**Definition 4 (Semantics).** Given a $\text{B}$-derivation $\mathcal{D}$ with conclusion $\square N, \cdot, \square N, N, \cdot, N \Rightarrow A$ 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 (all rules as typeset in Figure 2):

- If $\mathcal{D} = \text{id}$ then $f_{\mathcal{D}}(\cdot) := y$.
- If $\mathcal{D} = \text{w}_N(\mathcal{D}_0)$ then $f_{\mathcal{D}}(\vec{x}; \vec{y}) := f_{\mathcal{D}_0}(\vec{x}; \vec{y})$.
- If $\mathcal{D} = \text{w}_c(\mathcal{D}_0)$ then $f_{\mathcal{D}}(x; \vec{y}) := f_{\mathcal{D}_0}(x; \vec{y})$.
- If $\mathcal{D} = \text{e}_N(\mathcal{D}_0)$ then $f_{\mathcal{D}}(\vec{x}, y; \vec{y}, \vec{y}') := f_{\mathcal{D}_0}(\vec{x}, y; \vec{y}, \vec{y}')$.
- If $\mathcal{D} = \text{e}_c(\mathcal{D}_0)$ then $f_{\mathcal{D}}(x, x'; \vec{y}) := f_{\mathcal{D}_0}(x, x'; \vec{y})$.
- If $\mathcal{D} = \text{\textcircled{1}}(\mathcal{D}_0)$ then $f_{\mathcal{D}}(\vec{x}, x; \vec{y}) := f_{\mathcal{D}_0}(\vec{x}, x; \vec{y})$.
- If $\mathcal{D} = \text{\textcircled{r}}(\mathcal{D}_0)$ then $f_{\mathcal{D}}(\vec{x}; \cdot) := f_{\mathcal{D}_0}(\vec{x}; \cdot)$.
- If $\mathcal{D} = 0$ then $f_{\mathcal{D}}(\cdot) := 0$.
- If $\mathcal{D} = s_0(\mathcal{D}_0)$ then $f_{\mathcal{D}}(\vec{x}; \vec{y}) := s_0(\cdot; f_{\mathcal{D}_0}(\vec{x}; \vec{y}))$.
- If $\mathcal{D} = s_1(\mathcal{D}_0)$ then $f_{\mathcal{D}}(\vec{x}; \vec{y}) := s_1(\cdot; f_{\mathcal{D}_0}(\vec{x}; \vec{y}))$.
- If $\mathcal{D} = \text{cut}_N(\mathcal{D}_0, \mathcal{D}_1)$ then $f_{\mathcal{D}}(\vec{x}; \vec{y}) := f_{\mathcal{D}_0}(\vec{x}; \vec{y})$.
- If $\mathcal{D} = \text{cut}_c(\mathcal{D}_0, \mathcal{D}_1)$ then $f_{\mathcal{D}}(\vec{x}; \vec{y}) := f_{\mathcal{D}_0}(\vec{x}; \vec{y})$.
- If $\mathcal{D} = \text{cond}_N(\mathcal{D}_0, \mathcal{D}_1, \mathcal{D}_2)$ then:
  
  \[
  f_{\mathcal{D}}(\vec{x}; \vec{y}, 0) := f_{\mathcal{D}_0}(\vec{x}; \vec{y}) \\
  f_{\mathcal{D}}(\vec{x}; \vec{y}, \text{soy}) := f_{\mathcal{D}_0}(\vec{x}; \vec{y}, y) \text{ if } y \neq 0 \\
  f_{\mathcal{D}}(\vec{x}; \vec{y}, s_1y) := f_{\mathcal{D}_0}(\vec{x}; \vec{y}, y)
  \]
• If $\mathcal{D} = \text{cond}_e(\mathcal{D}_0, \mathcal{D}_1, \mathcal{D}_2)$ then:

\[
\begin{align*}
    f_D(0, x; y) &:= f_{\mathcal{D}_0}(x; y) \\
    f_D(s_0, x; y) &:= f_{\mathcal{D}_1}(x, x; y) \text{ if } x \neq 0 \\
    f_D(s_1, x; y) &:= f_{\mathcal{D}_2}(x, x; y)
\end{align*}
\]

• If $\mathcal{D} = \text{src}(\mathcal{D}_0, \mathcal{D}_1, \mathcal{D}_2)$ then:

\[
\begin{align*}
    f_D(0, x; y) &:= f_{\mathcal{D}_0}(x; y) \\
    f_D(s_0, x; y) &:= f_{\mathcal{D}_1}(x, x; y, f_D(x, x; y)) \text{ if } x \neq 0 \\
    f_D(s_1, x; y) &:= f_{\mathcal{D}_2}(x, x; y, f_D(x, x; y))
\end{align*}
\]

This formal semantics exposes how B-derivations and B functions relate. The rule src 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 $f_D$ is not quite defined according to function algebra B, due to the interpretation of the cut$_2$ 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 $\mathcal{D} : \square \Gamma, \tilde{N} \Rightarrow \square N$, there is a smaller B-derivation $\mathcal{D}' : \square \Gamma \Rightarrow \square N$ such that:

\[
f_{\mathcal{D}'}(x; y) = f_{\mathcal{D}}(x; y).
\]

**Proof.** The proof is by induction on the size of the derivation. The case where the last rule of $\mathcal{D}$ is an instance of id, $0, \sqcap, \text{cond}_N, \text{cond}_e$, or src are trivial. If the last rule of $\mathcal{D}$ is an instance of $e_N, e_G, \sqcup, s_i$, and w$_C$ 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 $\mathcal{D}'_0 : \square N, \ldots \Rightarrow \square N$ such that $f_{\mathcal{D}_0}(x; y) = f_{\mathcal{D}'_0}(x; y)$. Since $f_{\mathcal{D}'}(x; y) = f_{\mathcal{D}_0}(x; y) = f_{\mathcal{D}'_0}(x; y)$ 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 $f_{\mathcal{D}_1}(x; y) = f_{\mathcal{D}'_1}(x; y)$. Since $f_{\mathcal{D}_0}(x; y, f_{\mathcal{D}_1}(x; y)) = f_{\mathcal{D}'_0}(x; y)$, we set $\mathcal{D}' = \mathcal{D}'_0$. As for the case where the last rule is cut$_C$, by induction hypothesis, there exist derivations $\mathcal{D}'_0$ and $\mathcal{D}'_1$ such that $f_{\mathcal{D}_0}(x; y) = f_{\mathcal{D}'_0}(x; y)$ and $f_{\mathcal{D}_1}(x; y) = f_{\mathcal{D}'_1}(x; y)$, so that we define $\mathcal{D}'$ as the derivation obtained from $\mathcal{D}'_0$ and $\mathcal{D}'_1$ by applying the rule cut$_C$. \hfill \square

Our overloading of the notation $B$, for both a function algebra and for a type system, is now justified by:

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

**Proof.** Let us first prove the left-right implication by induction on $f \in B$. 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, $w_N$, $w_G$, $e_N$, and $e_G$. If $f = p(; x)$ then $\mathcal{D}$ is as follows:

\[
\begin{array}{c|c|c|c|c|c}
    \text{id} & \text{id} & \text{id} & \text{id} & \text{id} \\
    \text{cond}_N & 0 & N & \Rightarrow N & \Rightarrow N & \Rightarrow N \\
    \Rightarrow N & N & \Rightarrow N & \Rightarrow N & \Rightarrow N & \Rightarrow N
\end{array}
\]
If $f$ is $\text{cond}(x,y,z,w)$ then $D$ is constructed using id, $w_N$, $e_N$ and $\text{cond}_N$. Suppose now that $f(\bar{x};\bar{y}) = h(\bar{x},g(\bar{x});\bar{y})$. Then $D$ is as follows:

\[
\begin{array}{c}
\text{cut}_N \\
\square N \Rightarrow N \\
\text{cut}_N \\
\square N \Rightarrow \square N, \square N \Rightarrow N
\end{array}
\]

where $D_0$ and $D_1$ are such that $f_{D_0} = g$ and $f_{D_1} = h$. If $f(\bar{x};\bar{y}) = h(\bar{x},\bar{y},g(\bar{x});\bar{y}))$, then $D$ is as follows:

\[
\begin{array}{c}
\text{cut}_N \\
\square N, \square N \Rightarrow N \\
\text{cut}_N \\
\square N, \square N \Rightarrow N
\end{array}
\]

where $D_0$ and $D_1$ are such that $f_{D_0} = g$ and $f_{D_1} = h$. Last, suppose that $f(x,\bar{x},\bar{y})$ has been obtained by safe recursion from $g(\bar{x},\bar{y})$ and $h_i(x,\bar{x},\bar{y},y)$ with $i = 0, 1$. Then, $D$ is as follows:

\[
\begin{array}{c}
\text{cut}_N \\
\text{cut}_N \\
\text{cut}_N \\
\text{cut}_N \\
\square N, N \Rightarrow N
\end{array}
\]

where $D_0$, $D_1$, and $D_2$ are such that $f_{D_0} = g$, $f_{D_1} = h_0$ and $f_{D_2} = h_1$.

For the right-left implication, we prove by induction on the size of derivations that $D : \square N, \ldots, \square N, N, \ldots, N \Rightarrow C$ implies $f_D \in B$. The cases where the last rule of $D$ is id, $0$, $s_i$, $w_N$, $w_C$, $e_N$, $e_C$, $\square_l$, $\square_r$ are all straightforward using constants, successors and projections. If $D$ has been obtained from two derivations $D_0$ and $D_1$ by applying an instance of $\text{cut}_N$ then, by applying the induction hypothesis, $f_{D_0}(\bar{x};\bar{y}) = f_{D_0}(\bar{x};\bar{y})$. As for the case where the last rule is $\text{cut}_l$, by Proposition 5 there exists a derivation $D'_1$ with smaller size such that $f_{D'_1}(\bar{x};\bar{y}) = f_{D'_1}(\bar{x};\bar{y})$. By applying the induction hypothesis, we have $f_D(\bar{x};\bar{y}) = f_{D'_1}(\bar{x};\bar{y})$. If $D$ has been obtained from derivations $D_0$, $D_1$, $D_2$ by applying an instance of $\text{cond}_N$ then, by using the induction hypothesis, $f_{D_2}(\bar{x};\bar{y}) = \text{cond}(x, y, f_{D_0}(\bar{x};\bar{y}), f_{D_2}(\bar{x};\bar{y}), p(y)) \in B$. If the last rule is $\text{cond}_r$, by induction hypothesis, $f_{D_2}(x, \bar{y}) = \text{cond}(x, f_{D_0}(\bar{x};\bar{y}), f_{D_2}(p(x), \bar{x};\bar{y}), f_{D_2}(p(x), \bar{x};\bar{y}), p(y))$, where $p(x) = p(\pi_1^{\bar{y}}(x))$, and so $f_{D_2}(x, \bar{x}, \bar{y}) \in B$. Last, if $D$ has been obtained from derivations $D_0$, $D_1$, $D_2$ by applying an instance of $\text{sec}$, then $f_{D_2} \in B$ using the induction hypothesis and the safe recursion scheme.

dq

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

\section{Two-Sorted 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: $\text{CNB}$, which morally permits ‘nested’ versions of safe recursion, and $\text{CB}$, which will turn out to be closer to usual safe recursion.

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

7
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 \((\mathcal{B}^-)\)-coderivation \(\mathcal{D}\) is a possibly infinite rooted tree (of height \(\leq \omega\)) generated by the rules of \(\mathcal{B}^-\). Formally, we identify \(\mathcal{D}\) with a prefix-closed subset of \(\{0, 1, 2\}^*\) (i.e. a ternary tree) where each node is labelled by an inference step from \(\mathcal{B}^-\) such that, whenever \(\nu \in \mathcal{D}\) is labelled by a step \(S_1 \cdots S_n\), for \(n \leq 3\), \(\nu\) 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 \(\nu \in \{0, 1, 2\}^*\) are typically denoted \(\mathcal{D}_\nu\), so that \(\mathcal{D}_\epsilon = \mathcal{D}\). We write \(\nu \sqsubseteq \mu\) (or \(\nu \sqsubseteq \mu\)) if \(\nu\) is a prefix (respectively, a strict prefix) of \(\mu\), and in this case we say that \(\mu\) is above (respectively, strictly above) \(\nu\) or that \(\nu\) is below (respectively, strictly below) \(\mu\). 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.

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. Some examples of regular coderivations can be found in Figure 3, employing the following writing conventions:

**Convention 9 (Representing coderivations).** Henceforth, we may mark steps by • (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 \(\square N\) and \(\square w\) as well as certain structural steps, e.g. during a cut step.

Finally, when the sub-coderivations \(\mathcal{D}_0\) and \(\mathcal{D}_1\) above the second and the third premise of the conditional rule (from left) are similar (or identical), we may compress them into a single parametrised sub-coderivation \(\mathcal{D}_i\) (for \(i = 0, 1\)).

As discussed in [13, 15, 25], coderivations can be identified with Kleene-Herbrand-Gödel style equational programs, in general computing partial recursive functionals (see, e.g., [23, §63] for further details). We shall specialise this idea to our two-sorted setting.
Definition 10 (Semantics of coderivations). To each B⁻-coderivation \( D \) we associate a two-sorted Kleene-Herbrand-Gödel partial function \( f_D \) obtained by construing the semantics of Definition 4 as a (possibly infinite) equational program. Given a two-sorted function \( f(x,y) \), we say that \( f \) is defined by a B⁻-coderivation \( D \) if \( f_D(x,y) = f(x,y) \).

Remark 11. Note, in particular, that from a regular coderivation \( D \) we obtain a finite equational program determining \( f_D \). Of course, our overloading of the notation \( f_D \) is suggestive since it is consistent with that of Definition 4.

Example 12 (Regular coderivations, revisited). Let us consider the semantics of coderivations \( I, R \) and \( C \) from Figure 3.

- The partial functions \( f_I, f_R, f_C \) are given by the following equational program:
  \[
  f_I(x;_i) = f_I(f_I(x;);_i) \\
  f_R(x;_i) = f_R(x;_i) \\
  f_{I_{00}}(x) = f_{I_{00}}(x) \\
  f_{I_{01}}(x;_i) = s_1(x;_i) \\
  f_I(x;_i) = f_I(x;_i)
  \]

By purely equational reasoning, we can simplify this program to obtain \( f_I(x;_i) = f_I(s_1;_i;x;_i) \). Since the above equational program keeps increasing the input, the function \( f_I = f_I \) is always undefined.

- Let \( f_G(\bar{x},\bar{y}) \) and \( f_{H_i}(x,\bar{x},\bar{y}) (i = 0,1) \) be the functions defined by the regular B⁻-coderivations \( G \) and \( H_i \), respectively. Then the equational program for \( R \) can be rewritten as follows:

\[
\begin{align*}
  f_R(0,\bar{x};_i) &= f_G(\bar{x};_i) \\
  f_R(s_1x,\bar{x};_i) &= f_{H_i}(x,\bar{x},f_R(x,\bar{x};_i));_i)
\end{align*}
\]

which is an instance of a non-safe recursion scheme (on notation).

- The equational program of \( C \) can be written as:

\[
\begin{align*}
  f_C(z,0,0;_i) &= z \\
  f_C(z,s_1y,0;_i) &= s_1f_C(z,0;_i) \\
  f_C(x,y,0;_i) &= s_1f_C(0;_i,0,0;_i) \\
  f_C(z,s_1y,0;_i) &= s_1f_C(z,0;_i) \\
  f_C(x,y,0;_i) &= s_1f_C(0;_i,0,0;_i)
\end{align*}
\]

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

The above examples illustrate two undesirable features of regular B⁻-coderivations, from the point of view of implicit complexity:

I. on the one hand, despite being finitely presentable, they can define partial functions;
II. on the other hand, despite the presence of modalities implementing the normal/safe distinction of function arguments, they can define non-safe recursion schemes.

3.2 The progressing criterion

To address Problem I we shall adapt to our setting a well-known ‘termination criterion’ from non-wellfounded proof theory. First, let us recall some standard proof theoretic concepts about (co)derivations, similar to those in [13, 15, 25].

Definition 13 (Ancestry). Fix a coderivation \( D \). We say that a type occurrence \( A \) is an immediate ancestor of a type occurrence \( B \) in \( D \) 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 \( \Gamma \) or \( \Gamma' \), then furthermore they must be 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 $\sqcap N$ and infinitely often principal for a $\text{cond}_{\sqcap N}$ rule.

A coderivation is progressing if each of its infinite branches has a progressing thread.

**Example 15** (Regular coderivations, re-revisited). In Figure 3, $I$ has precisely one infinite branch (that loops on $\bullet$) which contains no instances of $\text{cond}_{\sqcap}$ at all. Therefore, $I$ is not progressing. On the other hand, $C$ has two simple loops, one on $\bullet$ and the other 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, $C$ is progressing. By the same reasoning, we can conclude that $R$ is progressing whenever $G$ and $H_i$ are.

Like in [13, 15, 25], the progressing criterion is sufficient to guarantee that the partial function computed is, in fact, a well-defined total function:

**Proposition 16.** If $D$ is progressing then $f_D$ is total.

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

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 $(\sqcap N^i)_{i \geq k}$ is progressing and so is infinitely often principal for $\text{cond}_{\sqcap}$, where the value of $n_i$ must strictly decrease (cf., again, Definition 4).

This contradicts the well-ordering property of the natural numbers. $\blacksquare$

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., [18] 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 [24].

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

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

$$f_{D'}(\bar{x}; \bar{y}) = f_D(\bar{x}; \bar{y}).$$
Proof sketch. By progressiveness, any infinite branch contains a cond\_□-step, which has non-modal succedent. Thus there is a set of cond\_□-occurrences that forms a bar across \( D \). By König Lemma, the set of all nodes of \( D \) below this bar, say \( X_D \), is finite. The proof now follows by induction on the cardinality of \( X_D \) 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.

3.3 Computational expressivity of coderivations

Problem II indicates that the modal/non-modal distinction for (progressing) \( B^- \)-coderivations, by itself, does not suffice to control complexity. Indeed it is not hard to see that, as it stands, this distinction is somewhat redundant for definable functions with only normal inputs. By inspection of Figure 2, we may safely replace each \( N \) by \( □N \) in any such coderivation, preserving progressiveness:

**Proposition 19.** Let \( D : □N, □N, □N, □N, □N, □N \Rightarrow N \) be a \( B^- \)-coderivation. Then, there exists a \( B^- \)-coderivation \( D^\square : □N, □N, □N, □N, □N \Rightarrow N \) s.t.:

- \( f_D(\bar{x}; \bar{y}) = f_D^\square(\bar{x}; \bar{y}); \)
- \( D^\square \) does not contain instances of \( w_N, e_N, \text{cut}_N, \text{cond}_N. \)

Moreover, \( D^\square \) is regular (resp. progressing) if \( D \) is.

Proof. We construct \( D^\square \) coinductively. The only interesting cases are when \( D \) is id or it is obtained from \( D_0 : \Gamma \Rightarrow N \) and \( D_1 : \Gamma, N \Rightarrow A \) by applying a \( \text{cut}_N \)-step. Then, \( D^\square \) is constructed, respectively, as follows:

\[
\begin{array}{c}
id \quad □N \Rightarrow N \\
\begin{array}{c}
□\Gamma \Rightarrow N \\
□\Gamma, □N \Rightarrow N
\end{array}
\end{array}
\]

Consequently, we may view (regular, progressing) \( B^- \)-coderivations as the type 0 (regular, progressing) fragment of the system \( CT \) from [13, 15, 25].

As a result we inherit the following characterisations:

**Proposition 20.** We have the following:

1. Any function \( f(\bar{x};) \) is defined by a progressing \( B^- \)-coderivation.
2. The class of regular \( B^- \)-coderivations is Turing-complete, i.e. they define every partial recursive function.
3. \( f(\bar{x};) \) is defined by a regular progressing \( B^- \)-coderivation if and only if \( f(\bar{x}) \) is type-1-primitive-recursive, i.e. it is in the level 1 fragment \( T_1 \) of Gödel’s \( \mathcal{T} \).

The proof of the above proposition can be found in Appendix A.

Given the computationally equivalent system \( CT_0 \) with contraction from [15], we can view the above result as a sort of ‘contraction admissibility’ for regular progressing \( B^- \)-coderivations. Call \( B^- + \{ e_N, c_{\geq N} \} \) the extension of \( B^- \) with
the rules $c_N$ and $c_{\Box N}$ below:

\[
\begin{align*}
\Gamma, N, N & \Rightarrow B & \Gamma, N & \Rightarrow B \\
\Gamma, N & \Rightarrow B & \Gamma, \Box N, \Box N & \Rightarrow B
\end{align*}
\]

(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.\(^1\) We have:

**Corollary 21.** $f(\hat{x}_i)$ is definable by a regular progressing $B^- + \{c_N, c_{\Box N}\}$-coderivation if and only if it is definable by a regular progressing $B^-$-coderivation.

### 3.4 Proof-level conditions motivated by implicit complexity

$B$-derivations locally introduce safe recursion by means of the rule $\text{srec}$, and Proposition 5 ensures that the composition schemes defined by the cut rules are safe. As suggested by Problem II, a different situation arises when we move to $B^-$-coderivations, where the lack of further constraints means that we can define ‘non-safe’ equational programs. We may recover safety by a natural proof-level condition:

**Definition 22 (Safety).** A $B^-$-coderivation is **safe** if each infinite branch crosses only finitely many cut-$\Box$-steps.

The corresponding equational programs of safe coderivations indeed only have safe inputs in hereditarily safe positions, as we shall soon see. Let us illustrate this by means of examples.

**Example 23.** The coderivation $I$ in Figure 3 is not safe, as there is an instance of cut-$\Box$ in the loop on $\bullet$, which means that there is an infinite branch crossing infinitely many cut-$\Box$-steps. By contrast, using the same reasoning, we can infer that the coderivation $C$ is safe. Finally, by inspecting the coderivation $R$ of Figure 3, we notice that the infinite branch that loops on $\bullet$ contains infinitely many cut-$\Box$ steps, so it too is not safe.

Perhaps surprisingly, however, the safety condition is not enough to restrict the set of $B^-$-definable functions to $\text{FPTIME}$, as the following example shows.

**Example 24 (Safe exponentiation).** Consider the following coderivation $E$,

\[
\begin{array}{l}
\text{id} & \quad \text{cond}_2 & \quad \vdots & \quad \text{cond}_2 & \quad \vdots \\
N & \Rightarrow N & \text{cut}_N & \Box N, N & \Rightarrow N & \Box N, N & \Rightarrow N \\
\Rightarrow & \text{cond}_2 & \quad & \Rightarrow & \quad & \Rightarrow \\
N & \Rightarrow N & \Box N, N & \Rightarrow N & \Box N, N & \Rightarrow N \\
\Rightarrow & \text{cut}_N & \quad & \Rightarrow & \quad & \Rightarrow \\
\Box N, N & \Rightarrow N & \Box N, N & \Rightarrow N & \Box N, N & \Rightarrow N \\
\Rightarrow & \text{cond}_2 & \quad & \Rightarrow & \quad & \Rightarrow \\
\end{array}
\]

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 $E$ has no instances of cut-$\Box$. Its associated equational program can be written as follows:

\[
\begin{align*}
& f_E(0; y) = s_0(; y) \\
& f_E(s_0x; y) = f_E(x; f_E(x; y)), \; x \neq 0 \\
& f_E(s_1x; y) = f_E(x; f_E(x; y))
\end{align*}
\]

(3)

The above equational program has already appeared in [22, 27]. It is not hard to show, by induction on $x$, that $f_E(x; y) = 2^{|x|} \cdot y$. Thus $f_E$ has exponential growth rate (as long as $y \neq 0$), despite being defined by a ‘safe’ recursion scheme.

\(^1\)Note that the totality argument of Proposition 16 still applies in the presence of these rules, cf. also [15].
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 circular proofs, and it is worth discussing.

**Remark 25 (On nesting and higher-order recursion).** As we have seen, namely in Proposition 20.3, (progressing) $B^\perp$-coderivations are able to simulate some sort of higher-order recursion, namely at type 1 (cf. also [15]). In this way it is arguably not so 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 ‘safe’ recursion operator:

$$\text{rec}_A : \square N \to (\square N \to A) \to A \to A$$

with $A = N \to N$, and $f(x) = \text{rec}_A(x, h, g)$ is defined as $f(0) = g$ and $f(s(x)) = h(x, f(x))$ for $x > 0$. By setting $g := \lambda y : N. s_0 y$ and $h := \lambda x : \square N. \lambda u : N. (\lambda y : N. u(u(y)))$ we easily check that $f_E(x; y) = \text{rec}_A(x, h, g)(y)$, where $E$ is as in Example 24. Hence, the function $f_E(x; y)$ can be defined by means of a higher-order version of safe recursion.

As noticed by Hofmann [22], and formally proved by Leivant [27], $\text{FELEMENTARY}$ can be characterised using higher-order safe recursion, thanks to this capacity to nest recursive calls. Moreover, Hofmann showed in [22] that by introducing a ‘linearity’ restriction on the operator $\text{rec}_A$, which prevents duplication of recursive calls, it is possible to recover the class $\text{FPTIME}$. The resulting type system, called $\text{SLR}$ (‘Safe Linear Recursion’), can thus be regarded as a higher-order formulation of $B$.

Following [22], 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 $f_E$ in Example 24 is due to the presence in $E$ 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. Again, we rather give a more natural proof-level criterion which does not depend on our intuitive notion of loop.

**Definition 26 (Left-leaning).** A $B^\perp$-coderivation is said to be left-leaning if each infinite branch goes right at a cut$_N$-step only finitely often.

**Example 27.** In Figure 3, $I$ is trivially left-leaning, as it contains no instances of cut$_N$ at all. The coderivations $C$ and $R$ are also left-leaning, since no infinite branch can go right at the cut$_N$ steps. By contrast, the coderivation $E$ in Example 24 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 present our circular systems:

**Definition 28 (Circular Implicit Systems).** $\text{CNB}$ is the class of safe regular progressing $B^\perp$-coderivations. $\text{CB}$ is the restriction of $\text{CNB}$ to only left-leaning coderivations. A two-sorted function $f(x; y)$ is $\text{CNB}$-definable (or $\text{CB}$-definable) if there is a coderivation $D \in \text{CNB}$ (resp., $D \in \text{CB}$) such that $f_{D}(x; y) = f(x; y)$.

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

**Proposition 29.** Let $D : \square \Gamma, \tilde{N} \Rightarrow \square N$ be a coderivation in $\text{CNB}$ (or $\text{CB}$). There exists a $\text{CNB}$-coderivation (resp., $\text{CB}$-coderivation) $D^* : \square \Gamma \Rightarrow \square N$ such that $f_{D^*}(x; y) = f_{D^*}(x; y)$.
3.5 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, since these conditions are defined at the proof-level rather than the thread-level, they are easy to check on regular coderivations:

**Proposition 30.** The safety and the left-leaning condition are \( \text{NL} \)-decidable for regular coderivations.

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

- the size of the certificate is smaller than the size of the description of \( G_D \);
- \( M \) 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. \( \square \)

The idea here is that, for regular coderivations, checking that no branch has infinitely many occurrences of a particular rule can be reduced to checking acyclicity of a certain subgraph, which is well-known to be in \( \text{coNL} = \text{NL} \).

Recall that progressiveness of regular coderivations is decidable by reduction to universality of Büchi automata, a \( \text{PSPACE} \)-complete problem. Indeed progressiveness itself is \( \text{PSPACE} \)-complete in many settings, cf. [31]. 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 31.** A safe \( B^- \)-coderivation is progressing iff every infinite branch has infinitely many \( \text{cond}_\square \)-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 \( D \). By safety, there exists a node \( v \) of \( B \) such that any sequent above \( v \) in \( B \) is not the conclusion of a \( \text{cut}_\square \)-step. Now, by inspecting the rules of \( B^- \) - \{ \text{cut}_\square \} we observe that:

- every modal formula occurrence in \( B \) has a unique thread along \( B \);
- infinite threads along \( B \) cannot start strictly above \( v \).

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

Thus, using similar reasoning to that of Proposition 30 we may conclude from Proposition 31 the following:

**Corollary 32.** Given a regular \( B^- \)-coderivation \( D \), the problem of deciding if \( D \) is in \( \text{CNB} \) (resp. \( \text{CB} \)) is in \( \text{NL} \).

Let us point out that the reduction above is similar to (and indeed generalises) an analogous one for cut-free extensions of Kleene algebra, cf. [17, Proposition 8].
4 SOME VARIANTS OF SAFE RECURSION

In this section we shall introduce various extensions of $B$ to ultimately classify the expressivity of the circular systems $CB$ and $CNB$. First, starting from the analysis of Example 24 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 4 summarises the function algebras considered and their relations.

4.1 Relativised algebras and nested recursion

One of the key features of the Bellantoni-Cook algebra $B$ is that ‘nesting’ of recursive calls is not permitted. For instance, let us recall the equational program from Example 24:

\[
\begin{align*}
\text{ex}(0; y) &= s_0 y \\
\text{ex}(s_i x; y) &= \text{ex}(x; \text{ex}(x; y))
\end{align*}
\]  (4)

Recall that $\text{ex}(x; y) = f_0^x(x; y) = 2^{2^{|x|}} \cdot y$. The “recursion step” on the second line is compatible with safe composition, in that safe inputs only occur in hereditarily safe positions, but one of the recursive calls takes another recursive call among its safe inputs. In Example 24 we showed how the above function $\text{ex}(x; y)$ can be $CNB$-defined. We thus seek a suitable extension of $B$ able to formalise such nested recursion to serve as a function algebraic counterpart to $CNB$.

It will be convenient for us 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 treat them as explicit functions too.

**Definition 33.** For all sets of oracles $\vec{a} = a_1, \ldots, a_k$, we define the algebra of $B^\vec{a}$ functions over $\vec{a}$ to include all the initial functions of $B^\vec{a}$ 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 $g(\vec{x}; \vec{y}), h(\vec{x}; \vec{y}, \vec{u})$ 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 $g(\vec{x}; \vec{y})$ over $\emptyset$ and $h(\vec{x}, \vec{y})$ over $\emptyset$ 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 $B^\vec{a}$ 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 $g(\vec{x}; \vec{y})$ is oracle-free. This restriction prevents oracles (and hence the recursive calls) appearing in normal position. The same condition on $h(\vec{x}; \vec{y})$ 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, $\lambda \vec{x}. f(\vec{x}; \vec{y})$ for the function taking only safe arguments $\vec{z}$ with $(\lambda \vec{x}. f(\vec{x}; \vec{y})) (\vec{z}; \vec{y}) = f(\vec{x}; \vec{y})$ (here $\vec{x}$ may be seen as parameters). Nested recursion can be formalised in the setting of algebras-with-oracles as follows:

**Definition 34 (Safe Nested Recursion).** We write $\text{snrec}$ for the scheme:

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

  
  \[
  f(0, \vec{x}; \vec{y}) = g(\vec{x}; \vec{y}) \\
  f(s_0 x, \vec{x}; \vec{y}) = h_0(\vec{x}. f(x; \vec{x}; \vec{y})) (x, \vec{x}; \vec{y}) \quad x \neq 0 \\
  f(s_1 x, \vec{x}; \vec{y}) = h_1(\vec{x}. f(x; \vec{x}; \vec{y})) (x, \vec{x}; \vec{y})
  \]

We write $\text{NB}(\vec{a})$ for the class of functions over $\vec{a}$ generated from $B^-$ under $\text{snrec}$ and Safe Composition (from Definition 33), and write simply $\text{NB}$ for $\text{NB}(\emptyset)$.

**Remark 35 (Safe Composition During Safe Recursion).** Note that Safe Nested Recursion also admits variants that are not morally ‘nested’ but rather use a form of ‘composition during recursion’:

- from $g(\vec{x}; \vec{y}), (\vec{g}_j(x; \vec{x}; \vec{y}))_{1 \leq j \leq k}$ and $h_i(x; \vec{x}; \vec{y}, (z_j)_{1 \leq j \leq k})$ define:

  \[
  f(0, \vec{x}; \vec{y}) = g(\vec{x}; \vec{y}) \\
  f(s_i x, \vec{x}; \vec{y}) = h_i(x, \vec{x}; \vec{y}, (f(x, \vec{x}; \vec{y}, (g_j(x; \vec{x}; \vec{y})))_{1 \leq j \leq k})
  \]

(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). In the conference version of this paper [11, p.8] we made an unsubstantiated claim that a function algebra extending $B^-$ by (5) above is equivalent to a restriction $\text{SB}$ of $\text{NB}$ requiring that oracles are *unnested* in Safe Composition: in item 1, one of $g(\vec{x}; \vec{y})$ and $h(\vec{x}; \vec{y}, y)$ must be oracle-free, i.e. over $\emptyset$. Such an equivalence is in fact not immediate, as our Safe Composition scheme allows substitution only along a single parameter, so there does not seem to be a direct way of expressing Equation (5) in $\text{NB}$ without nesting oracles, in the sense just described. While we did not rely on this equivalence for any of the main results of [11], in this work we simply omit reference to $\text{SB}$ and its variants to avoid any confusion.

### 4.2 Safe recursion on well-founded relations

Relativised function algebras may be readily extended by recursion on arbitrary well-founded relations. For instance, given a well-founded preorder $\triangleleft$, and writing $\triangleleft$ for its strict variant,$^2$ ‘safe recursion on $\triangleleft$’ is given by the 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 $\lambda v. \triangleleft x. f(v, \vec{x}; \vec{y})$ for a ‘guarded abstraction’. Formally:

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

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 $\triangleleft$.

---

$^2$To be precise, for a preorder $\triangleleft$ we write $x \triangleleft y$ if $x \triangleleft y$ and $y \not\triangleleft x$. As abuse of terminology, we say that $\triangleleft$ is well-founded just when $\triangleleft$ is.
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.

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

**Definition 36** (Permutations of prefixes). Let *n* denote {0, 1, ..., *n* − 1}. We write (x₀, x₁, ..., xₙ₋₁) ≤ (y₀, y₁, ..., yₙ₋₁) if, for some permutation *π* : [n] → [n], we have that *x*ₙ is a prefix of *y*ₙ, for all *i* < *n*. We write *x* ⊂ *y* if *x* ⊆ *y* but *y* ∉ *x*, i.e. there is a permutation *π* : [n] → [n] with *x*ₙ a prefix of *y*ₙ 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 ⊂ 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 ⊂:

**Definition 37** (Safe (nested) recursion on permutations of prefixes). We write NB*(¯a)* for the class of functions over ¯*a* generated from B under Safe Composition, the scheme snrec<sub>⊂</sub>,

- from *h*(*a*)(¯*x*; ¯*y*) over ¯*a*, define *f*(*x*; *y*) by:
  \[ *f*(*x*; *y*) = *h*(*λ* ¯*a* ⊂ ¯*x*, ¯*a* ⊂ *f*(¯*u*; ¯*v*))(*x*; *y*) \]

and the following generalisation of Safe Composition along a Normal Parameter:

(2)′ from *g*(¯*x*;) over ¯∅ and *h*(*x*, *y*; ¯*y*) over ¯*a* define *f*(*x*; ¯*y*) by:

\[ *f*(*x*; ¯*y*) = *h*(*x*; *y*; ¯*a*) = *h*(*x*; *g*(*x*;); *y*).

We define B*(¯a)* to be the restriction of NB*(¯a)* where every instance of snrec<sub>⊂</sub> has the form:

- from *h*(*a*)(¯*x*; ¯*y*) over ¯*a*, define *f*(*x*; *y*) by:
  \[ *f*(*x*; *y*) = *h*(*λ* ¯*a* ⊂ ¯*x*, ¯*a* ⊂ *f*(¯*u*; ¯*v*))(*x*; *y*) \]

We call this latter recursion scheme srec<sub>⊂</sub>, e.g. if we need to distinguish it from snrec<sub>⊂</sub>.

Note that the version of safe composition along a normal parameter above differs from the previous one, Item 2 from Definition 33, since the function *h* is allowed to use oracles. Again, this difference is inessential in terms of computational complexity, as we shall see. However, as we have mentioned, the greater expressivity of B<sup>⊂</sup> and NB<sup>⊂</sup> will facilitate our overall strategy for characterising CB and CNB, cf. Figure 1.

Let us take a moment to point out that NB*(¯a)* ⊇ B*(¯a)* indeed contain only well-defined total functions over the oracles ¯*a*, by reduction to induction on ⊂.

### 4.3 Simultaneous recursion schemes

For our main results, we will ultimately need a typical property of function algebras, that B<sup>⊂</sup> and NB<sup>⊂</sup> are closed under simultaneous versions of their recursion schemes.

**Definition 38** (Simultaneous schemes). We define schemes ssrec<sub>⊂</sub> and ssnrec<sub>⊂</sub>, respectively, as follows, for arbitrary ¯*a* = *a*₁, ..., *a*ₖ:

- from *h*₁(¯*a*)(¯*x*; ¯*y*) over ¯*a*, define *f*₁(¯*x*; ¯*y*) over ¯*b*, for 1 ≤ *i* ≤ *k*, by:
  \[ *f*₁(¯*x*; ¯*y*) = *h*₁((¯*a* ⊂ ¯*x*, ¯*a* ⊂ *f*(¯*u*; ¯*v*))₁≤₁≤*k*)(¯*x*; ¯*y*) \]
5 CHARACTERISATIONS FOR FUNCTION ALGEBRAS

In this section we characterise the complexities of the function algebras we introduced in the previous section. Namely, despite apparently extending $B$, $B^C$ still contains just the polynomial-time functions, whereas both $NB$ and $NB^C$ are shown to contain just the elementary functions. All such results rely on a ‘bounding lemma’ inspired by [6].
5.1 A relativised Bounding Lemma

Bellantoni and Cook showed in [6] that any function \( f(\vec{x}; \vec{y}) \in B \) satisfies the ‘poly-max bounding lemma’: there is a polynomial \( p_f(\vec{a}) \) such that:

\[
|f(\vec{x}; \vec{y})| \leq p_f(|\vec{x}|) + \max |\vec{y}|
\]  

(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 generalise the bounding lemma to a form suitable to the relativised 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. Both the statement and the proof are quite delicate 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 subtle.

To state and prove the Bounding Lemma, let us employ the notation \( \|\vec{x}\| = \sum |\vec{x}| \).

**Lemma 40 (Bounding Lemma).** Let \( f(\vec{a})(\vec{x}; \vec{y}) \in \text{NB}^c(\vec{a}) \), with \( \vec{a} = a_1, \ldots, a_k \). There is a function \( m^f(\vec{x}, \vec{y}) \) with,

\[
m^f(\vec{x}, \vec{y}) = e_f(\|\vec{x}\|) + d_f \sum c + \max |\vec{y}|
\]

for an elementary function \( e_f(n) \) and a constant \( d_f \geq 1 \), such that whenever there are constants \( \vec{c} = c_1, \ldots, c_k \) such that,

\[
|a_i(\vec{x}_i; \vec{y}_i)| \leq c_i + d_f \sum j \neq i c_j + \max |\vec{y}_i|
\]  

(8)

for \( 1 \leq i \leq k \), we have:

\[
|f(\vec{a})(\vec{x}; \vec{y})| \leq m^f(\vec{x}, \vec{y})
\]  

(9)

\[
f(\vec{a})(\vec{x}; \vec{y}) = f\left(\lambda \vec{x}_i, \lambda \vec{y}_i \leq m^f(\vec{x}, \vec{y}).a_i(\vec{x}_i; \vec{y}_i)\right)_i(\vec{x}; \vec{y})
\]  

(10)

Moreover, if in fact \( f(\vec{x}; \vec{y}) \in \text{B}^c(\vec{a}) \), 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 for the proof, namely in the case when \( f \) is defined by recursion, substituting different values for \( \vec{c} \) during an inductive argument.

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 fact, it is only important to distinguish whether \( d_f = 1 \) or not, since \( d_f \) forms the base of an exponent for defining \( e_f \) when \( f \) is defined by safe recursion.

Finally let us note that Equations (9) and (10) are somewhat dual: while the former bounds the output of a function (serving as a modulus of growth), the latter bounds the inputs (serving as a modulus of continuity).

**Remark 41.** 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 \( \vec{a} = \emptyset \) 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’).

\(^4\)Recall that, for \( \vec{x} = x_1, \ldots, x_n \), we write \( |\vec{x}| \) for \( |x_1|, \ldots, |x_n| \).

\(^5\)To be clear, here we write \( |\vec{y}_i| \leq m^f(\vec{x}, \vec{y}) \) here as an abbreviation for \( \{|y_{i_j}| : \leq m^f(\vec{x}, \vec{y})\} \).
Proof of Lemma 40. We prove Equation (9) and Equation (10) by induction on the definition of $f(\bar{x}; \bar{y})$, always assuming that we have $\bar{c}$ 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 $0, s, +, \times, x^y$ and projections. If $f(\bar{x}; \bar{y}) \in \mathcal{B}^C(\bar{a})$ 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.

Let us start with Equation (9). If $f(\bar{x}; \bar{y})$ is an initial function then it suffices to set $e_f(n) := 1 + n$ and $d_f := 1$.

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

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

$$|f(\bar{x}; \bar{y})| = |h(\bar{x}; g(\bar{x}); \bar{y})|$$

$$\leq e_h(|\bar{x}|) + d_h \sum \bar{c} + \max(|\bar{y}|, |g(\bar{x}); \bar{y})|$$

$$\leq e_h(|\bar{x}|) + d_h \sum \bar{c} + e_g(|\bar{x}|) + d_g \sum \bar{c} + \max |\bar{y}|$$

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 $f(\bar{x}; \bar{y})$ 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 $d_h \sum \bar{c}$ or the term $d_g \sum \bar{c}$ 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 $f(\bar{x}; \bar{y}) = h(\bar{x}; g(\bar{x}); \bar{y})$, 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 $|g(\bar{x}; |) \leq e_g(|\bar{x}|)$. We thus have,

$$|f(\bar{x}; \bar{y})| = |h(\bar{x}; g(\bar{x}); \bar{y})|$$

$$\leq e_h(|\bar{x}|, |g(\bar{x}); \bar{y}|) + d_h \sum \bar{c} + \max |\bar{y}|$$

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 $f(\bar{x}; \bar{y}) = h(\lambda \bar{u} \subset \bar{x}, \lambda \bar{o}, f(\bar{u}; \bar{z}))(\bar{x}; \bar{y})$, let $e_h, d_h$ be obtained from the inductive hypothesis. We claim that it suffices to set $d_f := d_h$ and $e_f(n) := nd_h^{n-1}e_h(n)$. 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 $n > 0$:

$$e_f(n) = nd_h^{n-1}e_h(n)$$

$$= d_h^n e_h(n) + (n - 1)d_h^{n-1}e_h(n)$$

$$= d_h^n e_h(n) + d_h(n - 1)d_h^{n-1}e_h(n)$$

$$\geq e_h(n) + d_h(n - 1)d_h^{n-1}e_h(n)$$

$$\geq e_h(n) + d_h(n - 1)d_h^{n-1}e_h(n - 1)$$

$$\geq e_h(n) + d_h e_f(n - 1)$$

Hence:

$$e_f(n) \geq e_h(n) + d_h e_f(n - 1) \hspace{1cm} (11)$$

Now, to show that Equation (9) bounds for the $f, e_f, d_f$ at hand, we proceed by a sub-induction on $|\bar{x}|$. For the base case, when $|\bar{x}| = 0$ (and so, indeed, $\bar{x} = \bar{0}$), note simply that $\lambda \bar{u} \subset \bar{x}, \lambda \bar{o}, f(\bar{u}; \bar{z})$ 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,

\[
|f(\bar{u}; \bar{v})| = |h(0)(\bar{u}; \bar{y})| \\
\leq e_h(0) + d_h \sum \bar{c} + \max |\bar{y}| \\
\leq e_f(0) + d_f \sum \bar{c} + \max |\bar{y}|
\]

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

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

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

\[
|f(\bar{u}; \bar{v})| = |h(\lambda \bar{u} \subset \bar{x}, \lambda \bar{u}.f(\bar{u}; \bar{v}))(\bar{x}; \bar{y})| \\
\leq e_h(\|\bar{x}\|) + d_h \sum \bar{c} + \max |\bar{y}| \quad \text{main IH} \\
\leq (e_h(\|\bar{x}\|) + d_he_f(\|\bar{x}\| - 1)) + \\
\quad + d_h \sum \bar{c} + \max |\bar{y}| \quad \text{def. of } c \\
\leq e_f(\|\bar{x}\|) + d_h \sum \bar{c} + \max |\bar{y}| \quad \text{(11)}
\]

Let us now prove Equation (10), and let \( e_f \) and \( d_f \) be constructed by induction on \( f \) as above. We proceed again by induction on the definition of \( f(\bar{a})(\bar{x}; \bar{y}) \), always making explicit the oracles of a function.

The initial functions and oracle calls are immediate, due to the ‘max \( |\bar{y}| \)’ term in Equation (10).

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

\[
\leq e_h(\|\bar{x}\|) + d_h \sum \bar{c} + \max(\|\bar{y}\|, |g(\bar{a})(\bar{x}; \bar{y})|) \\
\leq e_h(\|\bar{x}\|) + d_h \sum \bar{c} + e_g(\|\bar{x}\|) + d_g \sum \bar{c} + \max |\bar{y}| \quad \text{(8)} \\
\leq (e_h(\|\bar{x}\|) + e_g(\|\bar{x}\|)) + (d_h + d_g) \sum \bar{c} + \max |\bar{y}| \\
\leq e_f(\|\bar{x}\|) + d_f \sum \bar{c} + \max |\bar{y}|
\]

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

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

\[
\leq e_h(\|\bar{x}\| + |g(\bar{a})(\bar{x}; \bar{y})|) + d_h \sum \bar{c} + \max |\bar{y}| \\
\leq e_h(\|\bar{x}\| + e_g(\|\bar{x}\|)) + d_h \sum \bar{c} + \max |\bar{y}| \quad \text{Lemma 40} \\
\leq e_f(\|\bar{x}\|) + d_f \sum \bar{c} + \max |\bar{y}|
\]

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

Now, recall from the Bounding Lemma 40, whenever \( \bar{u} \subset \bar{x} \) (and so \( \|\bar{u}\| < \|\bar{x}\| \)), we have \( |f(\bar{u}; \bar{v})| \leq e_f(\|\bar{x}\| - 1) + d_f \sum \bar{c} + \max |\bar{v}| \). So by setting \( c = e_f(\|\bar{x}\| - 1) \) in the inductive hypothesis for \( h(\bar{u}, a) \), with \( a = \lambda \bar{u} \subset \bar{x}, \lambda \bar{u}.f(\bar{a})(\bar{u}; \bar{v}) \),
any oracle call from $h(\vec{a}, a)$ will only take safe inputs of lengths:
\[
\leq \epsilon_h(\|\vec{x}\|) + \epsilon_h(e_f(\vec{x}) - 1) + d_h \sum \vec{c} + \max |\vec{y}|
\leq e_f(\|\vec{x}\|) + d_f \sum \vec{c} + \max |\vec{y}|
\] (11)

This completes the proof. □

5.2 Soundness results

In this subsection we show that the function algebras $B^C$ and $NB^C$ (as well as NB) capture precisely the classes $FPTIME$ and $FELEMENTARY$, respectively. We start with $B^C$:

**Theorem 42.** Suppose $\vec{a}$ satisfies Equation (8) for some constants $\vec{c}$. We have the following:

1. If $f(\vec{x}; \vec{y}) \in B^C(\vec{a})$ then $f(\vec{x}; \vec{y}) \in FPTIME(\vec{a})$.
2. If $f(\vec{x}; \vec{y}) \in NB^C(\vec{a})$ then $f(\vec{x}; \vec{y}) \in FELEMENTARY(\vec{a})$.

Note in particular that, for $f(\vec{x}; \vec{y})$ in $B^C$ or $NB^C$, 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 Equation (10)) to replace certain oracle calls with explicit appropriately bounded functions computing their graphs.

From here we compute $f(\vec{x}; \vec{y})$ by a sort of `course-of-values' recursion on $\subseteq$, storing previous values in a lookup table. In the case of $B^C$, it is important that this table has polynomial-size, since there are only $m! \prod |\vec{x}|$ permutations of prefixes of a list $\vec{x} = x_1, \ldots, x_m$ (which is a polynomial of degree $m$).

**Proof of Theorem 42.** 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 $B^C(\vec{a})$.

Item 1 above, so that $e_f$ is a polynomial and $d_f = 1$.

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

\[ f(\vec{x}; \vec{y}) = h(\lambda(\vec{a}) \subseteq \vec{x}, \lambda(\vec{a}) \subseteq \vec{y}, f(\vec{u}; \vec{v}))(\vec{x}; \vec{y}) \]

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 40, and so $|f(\vec{x}; \vec{y})|$ is polynomial in $|\vec{x}, \vec{y}|$.

2. Second, note that the set $\{ (\vec{a}, \vec{b}) \mid \vec{a} \subseteq \vec{x}, \vec{b} \subseteq \vec{y} \}$ 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 $|x_1| \cdots |x_m| |y_1| \cdots |y_n| \leq |\vec{x}, \vec{y}|^{m+n}$ 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 $\subseteq \times \subseteq$ on $[\vec{x}; \vec{y}]$.

First, for convenience, temporarily extend $\subseteq \times \subseteq$ 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 $[\vec{x}; \vec{y}]$. 

22
Define \( F(\vec{x}, \vec{y}) := (f(\vec{u}, \vec{v}))_{\vec{u} \subseteq \vec{x}, \vec{v} \subseteq \vec{y}} \), i.e. it is the graph of \( \lambda \vec{u} \subseteq \vec{x}, \lambda \vec{v} \subseteq \vec{y}. f(\vec{u}, \vec{v}) \) 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:

\[
F(S(\vec{x}, \vec{y})) = \langle f(S(\vec{x}, \vec{y})), F(\vec{x}, \vec{y}) \rangle
\]

\[
= \langle h(F(\vec{x}, \vec{y}))(\vec{x}, \vec{y}), F(\vec{x}, \vec{y}) \rangle
\]

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 \( \text{NB}^C \) 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 (Equation (9)) 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

We are now ready to give our main function algebraic characterisation results for polynomial-time:

**Corollary 43.** The following are equivalent:

1. \( f(\vec{x}; \vec{y}) \in B \).
2. \( f(\vec{x}; \vec{y}) \in B^C \).
3. \( f(\vec{x}; \vec{y}) \in \text{FPTIME} \).

**Proof.** (1) \( \implies \) (2) is trivial, and (2) \( \implies \) (3) is given by Theorem 42.(1). Finally, (3) \( \implies \) (1) is from [6], stated in Theorem 2 earlier. □

The remainder of this subsection is devoted to establishing a similar characterisation for \( \text{NB}, \text{NB}^C \) and \( \text{FELEMENTARY} \).

To begin with, we recall the definition of the class \( \text{FELEMENTARY} \):

**Definition 44.** \( \text{FELEMENTARY} \) is the smallest set of functions containing:

- \( 0() := 0 \in \mathbb{N} \),
- \( \pi^n_j(x_1, \ldots, x_n) := x_j \), whenever \( 1 \leq j \leq n \);
- \( s(x) := x + 1 \);
- 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 \( f(\vec{x}; x), g(\vec{x}) \in \text{FELEMENTARY} \) then so is \( f(\vec{x}, g(\vec{x})) \);
- (Bounded recursion) If \( g(\vec{x}), h(x, \vec{x}, y), f(x, \vec{x}) \) are functions in \( \text{FELEMENTARY} \) 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 \( f(x, \vec{x}) \leq j(x, \vec{x}) \).

\(^6\)Here, as abuse of notation, we are now simply identifying \( F(\vec{x}; \vec{y}) \) with \( \lambda \vec{u} \subseteq \vec{x}, \lambda \vec{v} \subseteq \vec{y}. f(\vec{u}, \vec{v}) \).
Proposition 45 ([33]). Let \( f \in \textsc{FELEMENTARY} \) be a \( k \)-ary function. Then, there exists an integer \( m \) such that:
\[
f(\bar{x}) \leq E^m_2(\max(\bar{x}))
\]
where \( E^0_2(x) = x \) and \( E^{m+1}_2(x) = E_2(E^m_2(x)) \).

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

Definition 46. \( \textsc{FELEMENTARY}_{0,1} \) is the smallest set of functions containing:

- \( 0() := 0 \in \mathbb{N} \),
- \( \pi^m_i(x_1, \ldots, x_n) := x_j \), whenever \( 1 \leq j \leq n \);
- \( s_i(x) := 2x + i \), for \( i \in \{0, 1\} \);
- the function \( \epsilon(x, y) \) defined as follows:
  \[
  \epsilon(0, y) := s_0(y) \quad \epsilon(s_i x, y) := \epsilon(x, \epsilon(x, y))
  \]
and closed under the following:

- (Composition) If \( f(\bar{x}, x), g(\bar{x}) \in \textsc{FELEMENTARY}_{0,1} \) then so is \( f(\bar{x}, g(\bar{x})) \);
- (Bounded recursion on notation) If \( g(\bar{x}), h_i(\bar{x}, x, y), j(x, \bar{x}) \) are functions in \( \textsc{FELEMENTARY}_{0,1} \) then so is \( f(x, \bar{x}) \) given by:
  \[
  f(0, \bar{x}) := g(\bar{x}) \quad f(s_i x, \bar{x}) := h_i(x, \bar{x}, f(x, \bar{x}))
  \]
provided that \( f(x, \bar{x}) \leq j(x, \bar{x}) \).

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 \( \epsilon^n(x) \) as
\[
\epsilon^1(x) := \epsilon(x, 1) \quad \epsilon^{m+1}(x) := \epsilon^{m} (\epsilon^m(x))
\]
which allows us to prove that \( \epsilon^n(x) \) plays the role of rate growth function as the function \( E^m_2(x) \) (see Proposition 45).

Proposition 47.

1. \( \textsc{FELEMENTARY} = \textsc{FELEMENTARY}_{0,1} \);
2. for any \( f \in \textsc{FELEMENTARY}_{0,1} \) \( k \)-ary function there is an integer \( m \) such that:
\[
f(\bar{x}) \leq \epsilon^m(\max(\bar{x}))
\]

Proof. Let us first prove point 1. For the \( \supseteq \) direction we show that for any \( f \in \textsc{FELEMENTARY}_{0,1} \) there exists \( n \) such that:
\[
|f(\bar{x})| \leq 2^n \left( \sum |\bar{x}| \right)
\]
where \( 2_0(x) = x \) and \( 2_{n+1}(x) = 2^{2^n(x)} \). Since \( |x| = \lfloor \log_2(x + 1) \rfloor \), from the above inequation we would have that, for some \( m \):
\[
f(\bar{x}) \leq 2_{nm}(\sum \bar{x})
\]
which allows us to conclude \( f \in \textsc{FELEMENTARY} \), as the elementary time computable functions are exactly the elementary space ones. The inequation (13) can be proved by induction on \( f \), noticing that \( |\epsilon(x, y)| = 2|x| + |y| \). Concerning
the \subseteq direction, we prove by induction on \( f \in \text{FELEMENTARY} \) that there exists a function \( \hat{f} \in \text{FELEMENTARY}_{0,1} \) such that, for all \( \vec{x} = x_1, \ldots, x_n \):

\[
f(\vec{x}) = |\hat{f}(x_1, \ldots, x_n)|
\]

where \( m := s^n_i(0) = s_j(m.s_j(0)) \). Since the functions \(|\cdot|\) and \( x \mapsto x \) are both in \( \text{FELEMENTARY}_{0,1} \), we are able to conclude \( f \in \text{FELEMENTARY}_{0,1} \). The case \( f = 0 \) is trivial. As for the cases \( f = s \) and \( f = \pi^n \), we first notice that \(|\vec{x}| = x\). Then, we have:

\[
s(x) = |s(x)|
\]

\[
\pi^n_i(\vec{x}) = x_i = |x_i| = |\pi^n(x_1, \ldots, x_n)|
\]

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

\[
E^m_2(x) \leq \epsilon^{m+k}(x) \tag{14}
\]

where \( \epsilon^n(x) \) is as in (12). 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 \( \epsilon^k(x) \) 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:

\[
\hat{f}(\vec{x}) = h(\vec{x}, g(\vec{x}))
\]

\[
\hat{f}(\vec{x}) = |\hat{h}(x_1, \ldots, x_n, g(\vec{x}))|
\]

\[
\hat{f}(\vec{x}) = |\hat{h}(x_1, \ldots, x_n, \hat{g}(x_1, \ldots, x_n))|
\]

\[
\hat{f}(\vec{x}) = |\hat{f}(\vec{x})|
\]

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

\[
f(0, \vec{x}) = g(\vec{x})
\]

\[
f(y + 1, \vec{x}) = h(y, \vec{x}, f(y, \vec{x}))
\]

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

\[
\hat{f}(0, \vec{x}) = \hat{g}(\vec{x})
\]

\[
\hat{f}(s_1y, \vec{x}) = \hat{h}(y, \vec{x}, \hat{f}(y, \vec{x}))
\]

We show by induction on \( y \) that:

\[
f(y, \vec{x}) = |\hat{f}(y, x_1, \ldots, x_n)|
\]
We have:

\[\begin{align*}
f(0, \bar{x}) &= g(\bar{x}) \\
&= |\hat{g}(x_1, \ldots, x_n)| \\
&= |\hat{f}(0, x_1, \ldots, x_n)|
\end{align*}\]

\[\begin{align*}
f(y + 1, \bar{x}) &= h(y, \bar{x}, f(y, \bar{x})) \\
&= |\hat{h}(y, x_1, \ldots, x_n, f(y, \bar{x}))| \\
&= |\hat{h}(y, x_1, \ldots, x_n, |\hat{f}(y, x_1, \ldots, x_n)|)| \\
&= |\hat{h}(y, x_1, \ldots, x_n, f(y, x_1, \ldots, x_n))| \\
&= |\hat{f}(y + 1, x_1, \ldots, x_n)|
\end{align*}\]

since \(f(y, \bar{x}) \leq j(y, \bar{x})\) and \(j(y, \bar{x}) = |\hat{j}(y, x_1, \ldots, x_n)|\) by induction hypothesis, we are done.

Point 2 follows by point 1, Proposition 45 and (14).

Completeness for NB is based on a standard technique (see [6]), adapted to the case of \(\text{FELEMENTARY}\) in [35].

**Lemma 48.** For any \(f(\bar{x}) \in \text{FELEMENTARY}_{0,1}\) there are a function \(f^*(x; \bar{x}) \in \text{NB}\) and a monotone function \(t_f \in \text{FELEMENTARY}_{0,1}\) such that for all integers \(x\) and all \(w \geq t_f(\bar{x})\) we have \(f^*(w; \bar{x}) = f(\bar{x})\).

**Proof.** The proof is by induction on the definition of \(f\). If \(f\) is the zero, successor or projection function then \(f^* \in \text{NB}\). In this case we choose \(t_f = 0\). The function \(\varepsilon\) 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 \(\varepsilon\). 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 \(\varepsilon^* \in \text{NB}\) and \(t_\varepsilon \in \text{FELEMENTARY}_{0,1}\) with the required properties. If \(f(\bar{x}) = h(\bar{x}, g(\bar{x}))\) then we set \(f^*(w; \bar{x}) = h^*(w; \bar{x}, g^*(w; \bar{x}))\), which is in NB. Since the function \(g^*\) is clearly bounded by a monotone function \(b \in \text{FELEMENTARY}_{0,1}\), we set \(t_f(\bar{x}) = t_b(\bar{x}, b(\bar{x})) + t_b(\bar{x})\), which is monotone. By applying the induction hypothesis, if \(w \geq t_f(\bar{x})\) then:

\[f^*(w; \bar{x}) = h^*(w; \bar{x}, g^*(w; \bar{x})) = h^*(w; \bar{x}, g(\bar{x})) = h(\bar{x}, g(\bar{x}))\]

Let us finally suppose that \(f(x, \bar{y})\) is defined by bounded recursion on notation from \(g(\bar{y}), h_1(x, \bar{y}, f(x, \bar{y}))\) and \(j(x, \bar{y})\).

By applying the induction hypothesis we set:

\[\begin{align*}
\hat{f}(0, w; \bar{y}) &= g^*(w; \bar{y}) \\
\hat{f}(s_1(x), w; \bar{y}) &= \text{cond}(W(s_1(v), w; x), g^*(w; \bar{y}), h^*_1(w; W(u, w; x), \bar{y}, f^*(w, v; x, \bar{y})) \\
f^*(w, x, \bar{y}) &= \hat{f}(w, w; x, \bar{y})
\end{align*}\]

where \(W(u, w; x) = \hat{\cdot}(\hat{\cdot}(v; w); x)\) and \(\hat{\cdot}(x; y)\) is the truncated subtraction, which is in B, and hence in NB. We can easily show that \(f^*(w, x, \bar{y}) \in \text{NB}\). We define \(t_f(x, \bar{y}) = t_\varepsilon(\bar{y}) + \sum t_b(x, \bar{y}, j(x, \bar{y}))\), where \(j\) is the bounding function. Assuming \(j\) to be monotone, \(t_f\) is monotone too. We now show by induction on \(u\) that, whenever \(w \geq t_f(x, \bar{y})\) and \(w - x \leq u \leq w:\n
\[\hat{f}(u, w; x, \bar{y}) = f(x - (w - u), \bar{y})\]

If \(u = w - x\) then we have two cases:

- if \(u = 0\) then \(\hat{f}(0, w; x, \bar{y}) = g^*(w; \bar{y});\)
- if \(u = s_1(v)\) then, since \(W(s_1(v), w; x) = 0\), we have \(\hat{f}(s_1(v), w; x, \bar{y}) = g^*(w; \bar{y}).\)
Hence, in any case:

\[ \hat{f}(u, w; x, \bar{y}) = g'(w; \bar{y}) = g(\bar{y}) = f(0; \bar{y}) = f(x - (w - u), \bar{y}) \]

Let us now suppose that \( w - x < u \leq w \). This means that \( u = s_i(v) \) and \( W(s_i(v), w; x) > 0 \). Moreover, by monotonicity of \( t_f \) and definition of \( j \):

\[
\begin{align*}
w & \geq t_f(x, \bar{y}) \\
& \geq t_f(x - (w - s_i(v)), \bar{y}) \\
& \geq t_f(x - (w - v), \bar{y}) \\
& \geq t_{h_i}(x - (w - v), \bar{y}, j(x - (w - v), \bar{y})) \\
& \geq t_{h_i}(x - (w - v), \bar{y}, f(x - (w - v), \bar{y}))
\end{align*}
\]

By applying the induction hypothesis:

\[
\begin{align*}
\hat{f}(s_i(v), w; x, \bar{y}) &= h_i^*(w; W(v, w; x), \bar{y}, f(0, \bar{y})) \\
& = h_i^*(w; W(v, w; x), \bar{y}, f(x - (w - v), \bar{y})) \\
& = h_i^*(w; x - (w - v), \bar{y}, f(x - (w - v), \bar{y})) \\
& = h_i(x - (w - v), \bar{y}, f(x - (w - v), \bar{y})) \\
& = f(s_i((x - (w - v))), \bar{y}) \\
& = f((x - (w - s_i(v))), \bar{y})
\end{align*}
\]

Now, by (15), for all \( w \geq t_f(x, \bar{y}) \) we have:

\[ f^*(w; x, \bar{y}) = \hat{f}(w, w; x, \bar{y}) = f(x, \bar{y}) \]

and this concludes the proof.

\[ \square \]

**Theorem 49.** If \( f(\bar{x}) \in \text{FELEMENTARY} \) then \( f(\bar{x}; ) \in \text{NB} \).

**Proof.** First, given the function \( \text{ex}(x; y) \) in (4), we construct the function \( \text{ex}^m(x; ) \) by induction on \( m \):

\[
\begin{align*}
\text{ex}^0(x; ) &= \text{ex}(x; 1) \\
\text{ex}^{m+1}(x; ) &= \text{ex}^m(\text{ex}^1(x; ));
\end{align*}
\]

Hence \( \text{ex}^m(x) = \text{ex}^m(x; ) \), for all \( m \geq 1 \). Now, let \( f(\bar{x}) \in \text{FELEMENTARY} \). By Proposition 47.1, we have that \( f(\bar{x}) \in \text{FELEMENTARY}_{0,1} \). By Lemma 48, there exist \( f^*(w; \bar{x}) \in \text{NB} \) and a monotone function \( t_f \in \text{FELEMENTARY}_{0,1} \) such that, for all \( w, \bar{x} \) with \( w \geq t_f(\bar{x}) \), it holds that \( f^*(w; \bar{x}) = f(\bar{x}) \). By Proposition 47.2 there exists \( m \geq 1 \) such that:

\[ t_f(\bar{x}) \leq \text{ex}^m(\max(\bar{x}; )); \]

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

\[ f(\bar{x}; ) = f^*(\text{ex}^m(\max(\bar{x}; ); \bar{x}) \in \text{NB} \]

\[ \square \]

Now, by the same argument as for Corollary 43, only using Lemma 48 above instead of appealing to [6], we can give our main characterisation result for algebras for elementary computation:

**Corollary 50.** The following are equivalent:

1. \( f(\bar{x}; ) \in \text{NB} \).
(2) \( f(\bar{x};) \in NB \subset \).  
(3) \( f(\bar{x}) \in \text{FELEMENTARY} \).

6 CHARACTERISATIONS 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 that we have just established.

6.1 Completeness

To show that \( CB \) contains all polynomial-time functions, we may simply simulate Bellantoni and Cook’s algebra:

**Theorem 51.** If \( f(\bar{x}; \bar{y}) \in B \) then \( f(\bar{x} ; \bar{y}) \in CB \).

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

\[
\begin{align*}
\Gamma & \Rightarrow N, \quad \Box N, \Gamma, N \Rightarrow N \\
\Gamma & \Rightarrow N, \quad \Box N, \Gamma \Rightarrow N \\
\Delta & \Rightarrow N, \Gamma \Rightarrow N \\
\Delta & \Rightarrow N, \Gamma \Rightarrow N \\
\Delta & \Rightarrow N, \Gamma \Rightarrow N
\end{align*}
\]

We define \( D^* \) as follows:

\[
\begin{align*}
\Gamma & \Rightarrow N, \quad \Box N, \Gamma \Rightarrow N \\
\Gamma & \Rightarrow N, \quad \Box N, \Gamma \Rightarrow N \\
\Delta & \Rightarrow N, \Gamma \Rightarrow N \\
\Delta & \Rightarrow N, \Gamma \Rightarrow N
\end{align*}
\]

where \( \Gamma = \Box N, \Gamma \) and we identify the coderivations corresponding to the second and the third premise of the conditional rule, as they only differ on the sub-coderivation \( D_i^* \) (\( i = 1 \) for the former and \( i = 2 \) for the latter).

The above coderivation is clearly safe and left-leaning, by the inductive hypotheses for \( D_0, D_1, D_2 \). To see that it is progressing, note that any infinite branch is either eventually entirely in \( D_0^*, D_1^* \) or \( D_2^* \), 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 \( D \) can be is equivalent to:

\[
\begin{align*}
f_{D_0}(0, \bar{x}; \bar{y}) & \coloneqq f_{D_0^*}(\bar{x}; \bar{y}) \\
f_{D_1}(s0x, \bar{x}; \bar{y}) & \coloneqq f_{D_1^*}(x, \bar{x}; \bar{y}, f_{D_0^*}(x, \bar{x}; \bar{y})) \text{ if } x \neq 0 \\
f_{D_2}(s1x, \bar{x}; \bar{y}) & \coloneqq f_{D_2^*}(x, \bar{x}; \bar{y}, f_{D_0^*}(x, \bar{x}; \bar{y}))
\end{align*}
\]

so that \( f_D(\bar{x}; \bar{y}) = f_{D^*}(\bar{x}, \bar{y}) \). \( \square \)

We can also show that \( CNB \) is complete for elementary functions by simulating our nested algebra \( NB \). First, we need to introduce the notion of oracle for coderivations.
Definition 52 (Oracles for coderivations). Let $\bar{a} = a_1, \ldots, a_n$ be a set of safe-normal functions. A $B^- (\bar{a})$-coderivation is just a usual $B^-$-coderivation that may use initial sequents of the form $a_i \Gamma, N \Rightarrow N$, when $a_i$ takes $n_i$ normal and $m_i$ safe inputs. We write:

$$
\begin{align*}
\frac{a_i \Gamma, N \Rightarrow N}{\Gamma \Rightarrow A}
\end{align*}
$$

for a coderivation $D$ whose initial sequents are among the initial sequents $a_i \Gamma, N \Rightarrow N$, with $i = 1, \ldots, n$. We write $CNB(\bar{a})$ for the set of $CNB$-coderivations with initial functions $\bar{a}$. We may sometimes omit indicating some oracles $\bar{a}$ if it is clear from context.

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

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 $D(\bar{a}; \rho)$ from [15, Lemma 42], only we must give a more refined argument due to the unavailability of contraction in our system.

Lemma 53. Let $D(a)$ be a regular coderivation over initial sequents $\bar{a}, a$ of form:

$$
\begin{align*}
\frac{a \Delta \Rightarrow N \Gamma}{\Gamma \Rightarrow A}
\end{align*}
$$

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

$$
\begin{align*}
\frac{a^* \Gamma \Rightarrow N \Gamma}{\Gamma \Rightarrow A}
\end{align*}
$$

such that:

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

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

Proof Sketch. Let us consider a path $B$ from the root of $D(a)$ to an occurrence of the initial sequent $a$. Since the conclusion of $D(a)$ contains $k$ modal formulas, and $B$ cannot cross 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 cond_{\square_2}-step, we conclude that each of the $k \square N$-threads must end in the principal formula of a $w_{\square_2}$-step. For each $j$ we remove the corresponding $w_{\square_2}$-step in $B$ we add an extra $\square N$ 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 $D(a)$ to an initial sequent $a$ we obtain a coderivation with the desired properties. \hfill \Box

Finally we can give our main simulation result for CNB:

**Theorem 54.** If $f(\bar{x}; \bar{y}) \in \text{NB}$ then $f(\bar{x}; \bar{y}) \in \text{CNB}$.

**Proof.** We show by induction on $f(\bar{x}; \bar{y}) \in \text{NB}(\bar{a})$ that there is a $\text{CNB}(\bar{a})$-coderivation $D_f$ such that:

1. $f_{D_f}(\bar{x}; \bar{y}) = f(\bar{x}; \bar{y})$;
2. the path from the conclusion of $D_f$ to each initial sequent $a_i$ does not contain cut_{\square_2}-steps, $\square f$-steps and the leftmost premise of a cond_{\square_2}-step.

When $f(\bar{x}; \bar{y})$ is an initial function the definition of $D^*$ is straightforward, as $D = \emptyset$. If $f(\bar{x}; \bar{y}) = a_i(\bar{x}; \bar{y})$ then $D_f$ is the initial sequent $\square_{\text{NB}}$.

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

\[
\begin{array}{c}
\emptyset \\
\square \bar{N} \Rightarrow \quad \square \bar{N} \Rightarrow \quad \square \bar{N} \Rightarrow \\
\text{cut}_{\square_{\text{NB}}} \quad \text{cut}_{\square_{\text{NB}}} \quad \text{cut}_{\square_{\text{NB}}}
\end{array}
\]

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

\[
\begin{array}{c}
a_i \quad \square \bar{N}, \bar{N} \Rightarrow \quad a_i \quad \square \bar{N}, \bar{N} \Rightarrow \\
\text{cut}_{\square_{\text{NB}}} \quad \text{cut}_{\square_{\text{NB}}}
\end{array}
\]

Suppose that $f(x, x; y)$ is obtained from $g(x; y)$ over $\emptyset$, and $h(a)(x, x; y)$ over $a, a$ by $\text{snrec}$. Then, $f(0, x; y) = g(x; y)$ and $f(s_ja, x; y) = h(\lambda \bar{a} f(x, x; \bar{y}))(x, x; y)$. Note that $a$ has same type as $\lambda \bar{a} f(x, x; \bar{y})$, so it is a function taking
safe arguments only. Thus by induction hypothesis, $h(a)(x, \tilde{x}; \tilde{y})$ is NB$(\tilde{a}, a)$-defined by:

\[
\begin{align*}
\begin{array}{c}
\frac{a}{N \Rightarrow N} \\
\frac{a_i \triangle N, \tilde{N} \Rightarrow N}{D_h(a)}
\end{array}
\end{align*}
\]

where the path from the conclusion of $D_h(a)$ to each initial sequent $a_i$ does not contain cut$\square$-steps, $\square_l$-steps and the leftmost premise of a cond$\square$-step. By Lemma 53 we obtain the following coderivation:

\[
\begin{align*}
\begin{array}{c}
\frac{a^*}{\Box N, \triangle N, \tilde{N} \Rightarrow N} \\
\frac{a_i \triangle N, \triangle N, \tilde{N} \Rightarrow N}{D'_h(a^*)}
\end{array}
\end{align*}
\]

where $f_{D'_h(a^*)}(x, \tilde{x}; \tilde{y}) = f_{D_h(a(x, \tilde{x}))}(x, \tilde{x}; \tilde{y})$ and there exists a $\square N$-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 $D^*(a^*)$. We define $D_f$ as follows:

\[
\begin{align*}
\begin{array}{c}
\frac{\square_\circ}{\triangle N, \triangle \tilde{N}, \tilde{N} \Rightarrow N} \\
\frac{\square_\circ}{\square_\circ, \triangle N, \triangle \tilde{N}, \tilde{N} \Rightarrow N}
\end{array}
\end{align*}
\]

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 $\square N$, by the properties of $D^*_h$ inherited from Lemma 53, or $B$ shares a tail with an infinite branch of $D_g$ or $D'_h(a^*)$, which are progressing by the inductive hypotheses.
This completes the proof. □

6.2 The Translation Lemma

The goal of this section is to prove the following theorem, which shows a translation of CNB-coderivations into functions of NB$^C$ mapping, in particular, CB-derivations into functions of B$^C$:

**Theorem 55 (Translation Lemma).** Let $D$ be a CNB-coderivation. Then, there exists a set of $n$ functions $\{f_i\}_{1 \leq i \leq n}$ such that $f_1 = f_D$ and, for all $i$:

$$f_i(\bar{x}; \bar{y}) = h_i(\lambda \bar{u} \subset \bar{x}, \lambda \bar{v} f_j(\bar{u}; \bar{v}))_{1 \leq j \leq n}(\bar{x}; \bar{y})$$  \hspace{1cm} (16)

where $h_i \in \text{NB}^C(a_i)_{1 \leq i \leq n}$. Moreover, if $D$ is a CB-coderivation then, for all $i$:

$$f_i(\bar{x}; \bar{y}) = h_i(\lambda \bar{u} \subset \bar{x}, \lambda \bar{v} \leq \bar{y} f_j(\bar{u}; \bar{v}))_{1 \leq j \leq n}(\bar{x}; \bar{y})$$  \hspace{1cm} (17)

and $h_i \in \text{B}^C(a_i)_{1 \leq i \leq n}$.

This would conclude our characterisation of CB and CNB in terms of computational complexity.

We start with illustrating the overarching ideas behind its proof. We will then state and prove in more detail a stronger version of Theorem 55 (i.e., Lemma 60).

**Proof Idea.** By Definition 10 there exists a system of equations $S_D$ containing, for each node $v$ of $D$, an equation that defines the function $f_D$ in terms of the functions $f_{D_u}$ with $u$ immediately above $v$. Moreover, since $D$ is regular, by Remark 11 we may assume $S_D$ is a finite system of equations defining $f_D$ (i.e., $f_{D_u}$). It is then suggestive to represent

![Diagram of a definition tree and a cut tree]

Fig. 5. Structure of the definition tree of $D$.

We show that $D_f$ NB($\bar{a}$)-defines $f$ by induction on $x$. For the base case, $f_{D_f}^{(0, \bar{x}; \bar{y})} = f_D\bar{x}(\bar{x}; \bar{y}) = g(\bar{x}; \bar{y}) = f(0, \bar{x}; \bar{y})$. For the inductive step:

$$f_{D_f}^{(s_i x, \bar{x}; \bar{y})} = f_{D_f}^{(\lambda \bar{v}, f_{D_f}^{(x, \bar{x}; \bar{v})})(x, \bar{x}; \bar{y})} = f_{D_f}^{(\lambda \bar{v}, f(x, \bar{x}; \bar{v}))}(x, \bar{x}; \bar{y}) = h(\lambda \bar{v}, f(x, \bar{x}; \bar{v}))(x, \bar{x}; \bar{y}) = f(s_i x, \bar{x}; \bar{y}).$$

This completes the proof. □
\( \mathcal{D} \) as a 'definition tree', by replacing each node \( v \) with the corresponding function \( f_{D,v} \), as in Figure 5a, where each function in the tree is defined by one of the equations of \( S_D \). This representation of \( \mathcal{D} \) has the advantage of highlighting the inter-dependencies of the functions defined in \( S_D \).

We now discuss how the proof-theoretical properties of \( \mathcal{D} \) impose conditions on the equations of \( S_D \):

1. By safety of \( \mathcal{D} \), the tree in Figure 5a cannot contain the instance of \( \text{cut}_\mathcal{E} \) in Figure 5b. Hence, whenever \( S_D \) has functions \( f_{D,u} \) and \( f_{D,v} \) that are inter-dependent, no equation of \( S_D \) can define one of these functions by means of a composition along its normal parameters. By inspecting Definition 10 this implies that, whenever an equation in \( S_D \) defines a function \( f_{D,u} \) by affecting its normal parameters, \( u \) must be the conclusion of an instance of \( \text{cond}_\mathcal{E} \), i.e. this equation must be of the form \( f_{D,u} (s_1 x, \vec{z}, y) = f_{D,v} (x, \vec{z}, y) \), for some \( v \).

2. By progressiveness of \( \mathcal{D} \), there always exists an occurrence of \( \text{cond}_\mathcal{E} \) in-between two 'backpointers' \( b_i \), of Figure 5a.

3. If \( \mathcal{D} \) is left-leaning, then the tree in Figure 5a cannot contain the instance of \( \text{cut}_\mathcal{N} \) in Figure 5c. Hence, whenever the system of equations for \( \mathcal{D} \) contains two functions \( f_{D,u} \) and \( f_{D,v} \) that are inter-dependent, no equation can define one of these functions by means of a composition along its safe parameters. By inspecting Definition 10 this implies that, whenever an equation in \( S_D \) defines a function \( f_{D,u} \) by affecting its safe parameters, \( u \) must be the conclusion of an instance of \( \text{cond}_\mathcal{N} \), i.e. the equation must be of the form \( f_{D,u} (\vec{x}, \vec{y}, s_i z) = f_{D,v} (\vec{z}, \vec{y}, z) \), for some \( v \).

We can now simplify the equations in \( S_D \) to obtain a 'minimal' system of equations \( S^*_\mathcal{D} \), where each equation defines a function in \((f_i)_{1 \leq i \leq n}\), with \( f_i = f_{D,u} \) and \( f_i = f_{D,v} \) (see Figure 5a). More precisely, for all \( 1 \leq i \leq n \) there exists \( h_i \in \text{NB}^- (a_i)_{1 \leq i \leq n} \), for some oracles \((a_i)_{1 \leq i \leq n}\), such that the following equation is in \( S^*_\mathcal{D} \):

\[
 f_i (\vec{x}, \vec{y}) = h_i (\lambda \vec{u}. \lambda \vec{z}. f_j (\vec{u}; \vec{z}))_{1 \leq j \leq n} (\vec{x}, \vec{y})
\]

By point 1, each such equation can be rewritten as:

\[
 f_i (\vec{x}, \vec{y}) = h_i (\lambda \vec{u} \subseteq \vec{x}. \lambda \vec{z}. f_j (\vec{u}; \vec{z}))_{1 \leq j \leq n} (\vec{x}, \vec{y}) \tag{18}
\]

Moreover, by point 2 the relation \( \subseteq \) is strict (i.e. \( \vec{u} \subset \vec{x} \)) when \( j = i \). In particular, by point 3, if \( \mathcal{D} \) is left-leaning then \( h_i \in \text{B}^- (a_i)_{1 \leq i \leq n} \) and (18) above can be rewritten as:

\[
 f_i (\vec{x}, \vec{y}) = h_i (\lambda \vec{u} \subseteq \vec{x}. \lambda \vec{z} \subseteq \vec{y}. f_j (\vec{u}; \vec{z}))_{1 \leq j \leq n} (\vec{x}, \vec{y})
\]

It remains to show that the relation \( \subseteq \) is strict (i.e. \( \vec{u} \subset \vec{x} \)) when \( j \neq i \) in the above equations. This can be established by repeatedly applying the following operation for each equation in \( S^*_\mathcal{D} \) with shape (18): for all \( j \), if \( \subseteq \) is not strict in \( \lambda \vec{u} \subseteq \vec{x}. \lambda \vec{z} \subseteq \vec{y}. f_j (\vec{u}; \vec{z}) \) (i.e. \( \vec{u} \subset \vec{x} \)), replace \( f_j \) with its definition given by the corresponding equation of \( S^*_\mathcal{D} \). Such a procedure keeps 'unfolding' equations, and terminates as a consequence of point 2. □

We now introduce some technical notions and results required for proving Lemma 60.

First, we observe that a regular coderivation can be naturally seen as a finite tree with 'backpointers', a representation known as cycle normal form, cf. [9, 10].

**Definition 56** (Cycle normal form). Let \( \mathcal{D} \) be a regular \( \text{B}^- \) -coderivation. The cycle normal form (or simply cycle nf) of \( \mathcal{D} \) is a pair \((\mathcal{D}, R_\mathcal{D})\), where \( R_\mathcal{D} \) is a partial self-mapping on the nodes of \( \mathcal{D} \) whose domain of definition is denoted \( \text{Bud}(\mathcal{D}) \) and:

1. every infinite branch of \( \mathcal{D} \) contains some (unique) \( v \in \text{Bud}(\mathcal{D}) \);
2. if \( v \in \text{Bud}(\mathcal{D}) \) then both \( R_\mathcal{D}(v) \sqsubseteq v \) and \( \mathcal{D}_{R_\mathcal{D}(v)} = \mathcal{D}_v \);
(iii) for any two distinct nodes \( \mu \sqsubset v \) strictly below \( \text{Bud}(D) \), \( D_\mu \neq D_v \).

We call any \( v \in \text{Bud}(D) \) a *bud*, and \( R_D(v) \) its *companion*. A terminal node is either a leaf of \( D \) or a bud. The set of nodes of \( D \) bounded above by a terminal node is denoted \( T_D \). Given a node \( v \in T_D \), we define \( \text{Bud}_v(D) \) as the restriction of buds to those above \( v \).

**Remark 57.** The cycle normal form of a regular coderivation \( D \) always exists, as by definition any infinite branch contains a node \( v \) such that \( D_v = D_\mu \) for some node \( \mu \) below \( v \). \( \text{Bud}(D) \) is designed to consist of just the least such nodes, so that by construction the cycle normal form is unique. Note that \( \text{Bud}(D) \) must form an antichain: if \( \mu, v \in \text{Bud}(D) \) with \( \mu \sqsubset v \), then \( R_D(\mu) \sqsubset \mu \) are below \( \text{Bud}(D) \) but we have \( D_{R_D(\mu)} = D_\mu \) by (ii) above, contradicting the (iii).

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

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

**Proposition 58.** Let \( D \) be a regular \( B^- \)-coderivation with cycle nf \( \langle D, R_D \rangle \). For any \( v \in \text{Bud}(D) \), the (finite) path \( \pi \) from \( R_D(v) \) to \( v \) satisfies:

1. if \( D \) is progressing, \( \pi \) must contain the conclusion of an instance of \( \text{cond}_{2\mathbb{N}} \);
2. if \( D \) is a CNB-coderivation, \( \pi \) cannot contain the conclusion of \( \text{cut}_{2\mathbb{N}}, \Box, \forall, \land, \text{and the leftmost premise of } \text{cond}_{2\mathbb{N}} \);
3. if \( D \) is a CB-coderivation, \( \pi \) cannot contain the conclusion of \( \forall, \text{the leftmost premise of } \text{cond}_{\mathbb{N}}, \text{and the rightmost premise of } \text{cut}_{\mathbb{N}} \).

**Proof.** By definition of cycle nf, each path from \( R_D(v) \) to \( v \) in \( \langle D, R_D \rangle \) is contained in a branch of \( D \) such that each rule instance in the former appears infinitely many times in the latter. Hence:

(i) if \( D \) is progressing, the path contains the conclusion of an instance of \( \text{cond}_{2\mathbb{N}} \);
(ii) if \( D \) is safe, the path cannot contain the conclusion of a \( \text{cut}_{2\mathbb{N}} \) rule;
(iii) if \( D \) is left-leaning, the path cannot contain the rightmost premise of a \( \text{cut}_{\mathbb{N}} \) rule.

This shows point 1. Let us consider point 2. By point (ii), if \( D \) is safe then, going from a node \( \mu \) of the path to each of its children \( \mu' \), 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 \( \mu \) is the conclusion of \( \Box, \forall, \land, \text{or when } \mu' \text{ is the leftmost premise of } \text{cond}_{2\mathbb{N}} \). Since \( R_D(v) \) and \( v \) 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 \( D \) is safe and left-leaning then, going from a node \( \mu \) of the path to each of its children \( \mu' \), 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 \( \mu \) is the conclusion of \( \forall, \text{or when } \mu' \text{ is the leftmost premise of } \text{cond}_{\mathbb{N}} \). Since \( R_D(v) \) and \( v \) must be labelled with the same sequent, all such cases are impossible.

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

\[
\begin{array}{c}
\Gamma \Rightarrow A \\
\text{dis} \\
\Gamma \Rightarrow X
\end{array}
\]

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

To facilitate the translation, we shall define two disjoint sets $C_v$ and $O_v$. Intuitively, given a cycle $\langle D, R_D \rangle$ and $v \in T_D$, $C_v$ is the set of companions above $v$, while $O_v$ is the set of buds whose companion is strictly below $v$.

**Definition 59.** Let $\langle D, R_D \rangle$ be the cycle normal form of a $B^{-}\text{-coderivation}$ $D$. We define the following two sets for any $v \in T_D$:

$$C_v := \{\mu \in R_D(Bud_v(D)) \mid v \subseteq \mu\}$$

$$O_v := \{\mu \in Bud_v(D) \mid R_D(\mu) \subseteq v\}$$

We now state a generalised version of the translation lemma and prove it in all detail. The idea, here, is to analyse each node $v_0 \in T_D$ associating with it an instance of the scheme $\text{snrec}_C$ that simultaneously defines the functions \{$f_{D_v} \mid v \in C_v \cup \{v_0\}$\}, with the help of an additional set of oracles \{$f_{D_v} \mid \mu \in O_v$\}. When $v_0$ is the root of $D$, note that $O_{v_0} = \emptyset$ and so the function thus defined will be oracle-free. Thus we obtain an instance of $\text{snrec}_C$ defining $f_D$, and so $f_D \in B^C$ by Proposition 39.

**Lemma 60 (Translation Lemma, General Version).** Given $\langle D, R_D \rangle$ the cycle nf of a CNB-coderivation $D$, and $v_0 \in T_D$:

1. If $O_{v_0} = \emptyset$ then $f_{D_{v_0}} \in B^C\emptyset$. In particular, if $D$ is a CB-coderivation then $f_{D_{v_0}} \in B^C$.
2. If $O_{v_0} \neq \emptyset$ then $\forall v \in C_{v_0} \cup \{v_0\}$:

   $$f_{D_v}(\vec{x}; \vec{y}) = h_v((\lambda \vec{u} \subseteq \vec{x}, \lambda \vec{v}.f_{D_u}(\vec{u}; \vec{v}))_{\mu \in C_{v_0} \cup O_{v_0}})(\vec{x}; \vec{y})$$

   where:

   a. $h_v \in B^C((f_{D_\mu})_{\mu \in O_{v_0}}(a_\mu)_{\mu \in C_{v_0}})$ and hence $f_{D_v} \in B^C((f_{D_\mu})_{\mu \in O_{v_0}})$;

   b. the order $\vec{u} \subseteq \vec{x}$ is strict if either $v, \mu \in C_{v_0}$ or the path from $v$ to $\mu$ in $D_{v_0}$ contains the conclusion of an instance of $\text{cond}_C$;

   c. if $D$ is a CB-coderivation then $\vec{u} \subseteq \vec{y}$.

**Proof.** Points 1 and 2 are proven by simultaneous induction on the longest distance of $v_0$ from a leaf of $T_D$. Notice that in the following situations only point 1 applies:

- $v$ is the conclusion of an instance of $\text{id}$ or $\emptyset$;

- $v$ is the conclusion of an instance of $w_N$, $\Box_D$, and $\text{cut}_{\Box_N}$;

- $v$ is the conclusion of an instance of $w_N$ and $D$ is a CB-coderivation.

In particular, the last two cases hold by Proposition 58.2-3, as it must be that $O_v = \emptyset$ for any premise $v'$ of $v_0$. Let us discuss the case where $v_0$ is the conclusion of a cut$_C$ rule with premises $v_1$ and $v_2$. By induction on point 2 we have $f_{D_{v_0}}(\vec{x}; \vec{y}), f_{D_{v_0}}(\vec{x}; \vec{y}) \in B^C$. Since the conclusion of $D_{v_0}$ has modal succedent, by Proposition 18 there must be a coderivation $D^*$ such that $f_{D^*}(\vec{x}) = f_{D^*}(\vec{x}; \vec{y}) \in B^C$. Moreover, by Proposition 29, if $D_{v_0}$ is a CB-coderivation then $D^*$ is. Hence, we define $f_{D_{v_0}}(\vec{x}; \vec{y}) = f_{D_{v_0}}(\vec{x}, f_{D^*}(\vec{x}; \vec{y}) \in B^C$. If moreover $D$ is a CB-coderivation then, by applying the induction hypothesis, we obtain $f_{D_{v_0}} \in B^C$.

Let us now consider point 2. If $v_0$ is the conclusion of a bud then $O_{v_0} = \{v_0\}$, $C_{v_0} = \emptyset$, and all points hold trivially. The cases where $v_0$ is an instance of $w_N$, $e_N$, $\Box_D$, $\text{cut}_s$, $s_0$ or $s_1$ are straightforward. Suppose that $v_0$ is the conclusion of a cond$_C$ step with premises $v', v_1$, and $v_2$, and let us assume $O_{v'} \neq \emptyset$, $O_{v_1} \neq \emptyset$. By Proposition 58.2 we have $O_{v_2} = \emptyset$, so
that \(f_{D'} \in NB^C\) by induction hypothesis on point 1. By definition, \(O_{v_1} = O_{v_1} \cup O_{v_2}\) and \(C_{v_1} = C_{v_1} \cup C_{v_2}\). Then, we set:

\[
f_{D_{v_1}}(x, \bar{x}; \bar{y}) = \text{cond}(x, f_{D_{v_1}}(\bar{x}; \bar{y}), f_{D_{v_1}}(p(x), \bar{x}; \bar{y}), f_{D_{v_2}}(p(x), \bar{x}; \bar{y}))
\]

where \(p(x)\) can be defined from \(p(x)\) and projections. By induction hypothesis on \(v_i\):

\[
f_{D_{v_i}}(p(x), \bar{x}; \bar{y}) = \delta_{v_i}((\lambda u, \bar{u} \subseteq p(x), \bar{x}, \lambda \bar{a}, f_{D_p}(u, \bar{u}; \bar{v}))_{\mu \in C_{v_i} \cup O_{v_i}})(p(x), \bar{x}; \bar{y})
\]

\[
h_{v_i}((\lambda u, \bar{u} \subseteq x, \bar{x}, \lambda \bar{a}, f_{D_p}(u, \bar{u}; \bar{v}))_{\mu \in C_{v_i} \cup O_{v_i}})(p(x), \bar{x}; \bar{y})
\]

hence

\[
f_{D_{v_1}}(x, \bar{x}; \bar{y}) = h_{v_1}((\lambda u, \bar{u} \subseteq x, \bar{x}, \lambda \bar{a}, f_{D_p}(u, \bar{u}; \bar{v}))_{\mu \in C_{v_1} \cup O_{v_1}})(x; \bar{x}; \bar{y})
\]

for some \(h_{v_1}\). By IH, \(h_{v_1} \in NB^C((f_{D_{v_1}})_{\mu \in C_{v_1} \cup O_{v_i}})\) and \(f_{D_{v_2}} \in NB^C((f_{D_{v_2}})_{\mu \in C_{v_2}})\). This shows point 2a. Point 2b is trivial, and point 2c holds by applying the induction hypothesis.

Let us now consider the case where \(v_0\) is an instance of \(\text{cond}_X\), assuming \(O_{\bar{v}} \neq \emptyset\) and \(O_{v_2} \neq \emptyset\). The only interesting case is when \(D\) is a CB-coderivation. By Proposition 58.3 we have \(O_{v'} = \emptyset\), so that \(f_{D_{v'}} \in B^C\) by induction hypothesis on point 1. By definition, \(O_{v_1} = O_{v_1} \cup O_{v_2}\) and \(C_{v_1} = C_{v_1} \cup C_{v_2}\). Then, we set:

\[
f_{D_{v_1}}(x, \bar{x}; y, \bar{y}) = \text{cond}(x, y, f_{D_{v'}}(\bar{x}, \bar{y}), f_{D_{v_1}}(p(x), y, \bar{y}), f_{D_{v_2}}(\bar{x}, p(x), \bar{y}))
\]

By induction hypothesis on \(v_i\):

\[
f_{D_{v_i}}(x, \bar{x}; p(y), \bar{y}) = h_{v_i}((\lambda u, \bar{u} \subseteq x, \bar{x}, \lambda \bar{a}, f_{D_p}(u, \bar{u}; \bar{v}))_{\mu \in C_{v_i} \cup O_{v_i}})(x, \bar{x}; y, \bar{y})
\]

\[
h_{v_i}((\lambda u, \bar{u} \subseteq x, \bar{x}, \lambda \bar{a}, f_{D_p}(u, \bar{u}; \bar{v}))_{\mu \in C_{v_i} \cup O_{v_i}})(x, \bar{x}; y, \bar{y})
\]

where \(\bar{z} = p(y), \bar{y}\), and hence

\[
f_{D_{v_1}}(x, \bar{x}; y, \bar{y}) = h_{v_1}((\lambda u, \bar{u} \subseteq x, \bar{x}, \lambda \bar{a}, f_{D_p}(u, \bar{u}; \bar{v}))_{\mu \in C_{v_1} \cup O_{v_1}})(x, \bar{x}; y, \bar{y})
\]

for some \(h_{v_1}\). This shows point 2c. Point 2a and 2b are given by the induction hypothesis.

Let us now consider the case where \(v_0\) is the conclusion of an instance of \(\text{dis}\) with premise \(v'\), where \(X\) is the set of nodes labelling the rule. We have \(O_{v_1} = O_{v'} + X\) and \(C_{v_1} = C_{v'} + \{v_0\}\). We want to find \((h_{v'})_{v' \in C_{v'} \cup \{v_0\}}\) defining the equations for \((f_{D_{v'}})_{\mu \in C_{v'} \cup \{v_0\}}\) in such a way that points 2a-2c hold. We shall start by defining \(h_{v_1}\). First, note that, by definition of cycle \(f_{D_{v_1}}(x, \bar{x}; \bar{y}) = f_{D_{v'}}(x, \bar{x}; \bar{y}) = f_{D_{v_2}}(x, \bar{x}; \bar{y})\) for all \(\mu \in X\). By induction hypothesis on \(v'\) there exists a family \((g_{v'})_{v' \in C_{v'} \cup \{v_0\}}\) such that:

\[
f_{D_{v'}}(x, \bar{x}; \bar{y}) = g_{v'}((\lambda u, \bar{u} \subseteq x, \lambda \bar{a}, f_{D_p}(u, \bar{u}; \bar{v}))_{\mu \in C_{v'} \cup O_{v'}})(x, \bar{x}; \bar{y})
\]

and, moreover, for all \(v \in C_{v'}\):

\[
f_{D_{v}}(x, \bar{x}; \bar{y}) = g_{v'}((\lambda u, \bar{u} \subseteq x, \lambda \bar{a}, f_{D_p}(u, \bar{u}; \bar{v}))_{\mu \in C_{v'} \cup O_{v'}})(x, \bar{x}; \bar{y})
\]
Since $O_{\nu'} = O_\nu \cup X$ and the path from $v'$ to any $\mu \in X$ must cross an instance of $\text{cond}_{\Omega}$ by Proposition 58.1, the induction hypothesis on $v'$ (point 2b) allows us to rewrite (19) as follows:

$$f_{D_{\nu'}}(\bar{x}; \bar{y}) = g_{v'}((\lambda \bar{u} \subseteq \bar{x}, \lambda \bar{v}, f_{D_{\nu'}}(\bar{u}; \bar{v}))_{v \in C_{\nu'}}; (\lambda \bar{u} \subseteq \bar{x}, \lambda \bar{v}, f_{D_{\nu'}}(\bar{u}; \bar{v}))_{\mu \in O_{\nu'}}; (\lambda \bar{u} \subseteq \bar{x}, \lambda \bar{v}, f_{D_{\nu}}(\bar{u}; \bar{v}))_{\mu \in X})(\bar{x}; \bar{y})$$

(21)

On the other hand, for all $v \in C_{\nu'}$, for all $\bar{u} \subseteq \bar{x}$ and $\bar{v}$, the equation in (20) can be rewritten as:

$$f_{D_{\nu}}(\bar{u}; \bar{v}) = g_v((\lambda \bar{w} \subseteq \bar{u}, \lambda \bar{v}, f_{D_{\nu}}(\bar{w}; \bar{v}'))_{\mu \in C_{\nu'}}, (\lambda \bar{w} \subseteq \bar{u}, \lambda \bar{v}, f_{D_{\nu}}(\bar{w}; \bar{v}'))_{\mu \in O_{\nu'}}, (\lambda \bar{w} \subseteq \bar{u}, \lambda \bar{v}, f_{D_{\nu}}(\bar{w}; \bar{v}'))_{\mu \in X})(\bar{u}; \bar{v})$$

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

$$\lambda \hat{u} \subseteq \bar{x}, \lambda \hat{v}, f_{D_{\nu}}(\bar{u}; \bar{v})$$

$$\lambda \hat{u} \subseteq \bar{x}, \lambda \hat{v}, f_{D_{\nu}}(\bar{u}; \bar{v})$$

(22)

Now, since the paths from $v'$ to any $\mu \in X$ in $D$ must contain an instance of $\text{cond}_{\Omega}$, for all $v \in C_{\nu'}$ and all $\mu \in X$, we have that either the path from $v'$ to $\nu$ contains an instance of $\text{cond}_{\Omega}$ or the path from $v$ to $\mu$ does. By applying the induction hypothesis on $v'$ (point 2b), given $v \in C_{\nu'}$ and $\mu \in X$, either $\lambda \hat{u} \subseteq \bar{x}, \lambda \hat{v}, f_{D_{\nu}}(\bar{u}; \bar{v})$ in (21) is such that $\bar{u} \subseteq \bar{x}$, or $\lambda \hat{w} \subseteq \bar{u}, \lambda \hat{v}, f_{D_{\nu}}(\bar{w}; \bar{v}')$ in (22) is such that $\bar{w} \subseteq \bar{u}$. This means that, for any $\mu \in X$, $\lambda \hat{w} \subseteq \bar{x}, \lambda \hat{v}, f_{D_{\nu}}(\bar{w}; \bar{v}')$ in (23) is such that $\bar{w} \subseteq \bar{x}$. For each $v \in C_{\nu'}$, by rewriting $\lambda \hat{u} \subseteq \bar{x}, \lambda \hat{v}, f_{D_{\nu}}(\bar{u}; \bar{v})$ in (21) according to the equation in (23) we obtain:

$$f_{D_{\nu}}(\bar{x}; \bar{y}) = t_v((\lambda \bar{u} \subseteq \bar{x}, \lambda \bar{v}, f_{D_{\nu}}(\bar{u}; \bar{v}))_{\mu \in C_{\nu'} \cup \{\nu\}}; (\lambda \bar{u} \subseteq \bar{x}, \lambda \bar{v}, f_{D_{\nu}}(\bar{u}; \bar{v}))_{\mu \in O_{\nu'}}; (\lambda \bar{u} \subseteq \bar{x}, \lambda \bar{v}, f_{D_{\nu}}(\bar{u}; \bar{v}))_{\mu \in X})(\bar{x}; \bar{y})$$

(23)

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

$$f_{D_{\nu_0}}(\bar{x}; \bar{y}) = h_{\nu_0}((\lambda \bar{u} \subseteq \bar{x}, \lambda \bar{v}, f_{D_{\nu}}(\bar{u}; \bar{v}))_{\mu \in C_{\nu} \cup \{\nu\}})(\bar{x}; \bar{y})$$

(24)

which satisfies point 2b. From (24) we are able to find the functions $(h_v)_{v \in C_{\nu'}}$ defining the equations for $(f_{D_{\nu}})_{v \in C_{\nu'}}$. Indeed, the induction hypothesis on $v'$ gives us (20) for any $v \in C_{\nu'}$. We rewrite in each such equation any $\lambda \hat{u} \subseteq \bar{x}, \lambda \hat{v}, f_{D_{\nu}}(\bar{u}; \bar{v})$ such that $\mu \in X$ according to equation (24), as $f_{D_{\nu_0}} = f_{D_{\nu}}$ for any $\mu \in X$. We obtain the following equation for any $v \in C_{\nu'}$:

$$f_{D_{\nu}}(\bar{x}; \bar{y}) = t_v((\lambda \bar{u} \subseteq \bar{x}, \lambda \bar{v}, f_{D_{\nu}}(\bar{u}; \bar{v}))_{\mu \in C_{\nu} \cup \{\nu\}}; (\lambda \bar{u} \subseteq \bar{x}, \lambda \bar{v}, f_{D_{\nu}}(\bar{u}; \bar{v}))_{\mu \in O_{\nu'}}; (\lambda \bar{u} \subseteq \bar{x}, \lambda \bar{v}, f_{D_{\nu}}(\bar{u}; \bar{v}))_{\mu \in X})(\bar{x}; \bar{y})$$

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

$$f_{D_{\nu}}(\bar{x}; \bar{y}) = h_v((\lambda \bar{u} \subseteq \bar{x}, \lambda \bar{v}, f_{D_{\nu}}(\bar{u}; \bar{v}))_{\mu \in C_{\nu} \cup \{\nu\}})(\bar{x}; \bar{y})$$

(25)

It remains to show that (24) and (25) satisfy points 2a and 2c. Concerning point 2a, on the one hand for all $v \in C_{\nu_0}$ we have $h_v \in \text{NB}_C((f_{D_{\nu}})_{\mu \in O_{\nu'}}, (a_\mu)_{\mu \in C_{\nu'}})$, with $(a_\mu)_{\mu \in C_{\nu'}}$ oracle functions. On the other hand, by applying the induction hypothesis, we have $f_{D_{\nu_0}} = f_{D_{\nu}} \in \text{NB}_C((f_{D_{\nu}})_{\mu \in O_{\nu'}})$ and $f_{D_{\nu}} \in \text{NB}_C((f_{D_{\nu}})_{\mu \in O_{\nu'}})$, for all $v \in C_{\nu_0}$. Since $O_{\nu'} = O_{\nu} \cup X$ and $f_{D_{\nu}} = f_{D_{\nu}}$ for all $\mu \in X$, we have both $f_{D_{\nu_0}} \in \text{NB}_C((f_{D_{\nu}})_{\mu \in O_{\nu'}})$ and $f_{D_{\nu}} \in \text{NB}_C((f_{D_{\nu}})_{\mu \in O_{\nu'}})$, for all $v \in C_{\nu_0}$, and hence $f_{D_{\nu}} \in \text{NB}_C((f_{D_{\nu}})_{\mu \in O_{\nu'}})$, for all $v \in C_{\nu_0}$. Point 2c follows by applying the induction hypothesis, as the construction does not affect the safe arguments.
Last, suppose that \( v_0 \) is the conclusion of an instance of \( \text{cut}_N \) with premises \( v_1 \) and \( v_2 \). We shall only consider the case where \( O_{v_i} \neq \emptyset \) for \( i = 1, 2 \). Then, \( O_{v_1} = O_{v_1} \cup O_{v_2} \) and \( C_{v_1} = C_{v_1} \cup C_{v_2} \). We have:

\[
f_{D_{v_0}}(\bar{x}; \bar{y}) = f_{D_{v_2}}(\bar{x}; f_{D_{v_1}}(\bar{x}; \bar{y}), \bar{y})
\]

By induction hypothesis on \( v_1 \):

\[
f_{D_{v_1}}(\bar{x}; \bar{y}) = h(v_1)((\lambda \bar{u} \subseteq \bar{x}, \lambda \bar{v}. f_{D_{v_2}}(\bar{u}; \bar{v}))_\mu \in C_{v_1} \cup O_{v_1})(\bar{x}; \bar{y})
\]

Moreover, induction hypothesis on \( v_2 \):

\[
f_{D_{v_2}}(\bar{x}; \bar{y}, \bar{y}) = h(v_2)((\lambda \bar{u} \subseteq \bar{x}, \lambda \bar{v}. f_{D_{v_2}}(\bar{u}; \bar{v}))_\mu \in C_{v_2} \cup O_{v_2})(\bar{x}; \bar{y}, \bar{y})
\]

So that:

\[
f_{D_{v_0}}(\bar{x}; \bar{y}) = h(v_0)((\lambda \bar{u} \subseteq \bar{x}, \lambda \bar{v}. f_{D_{v_2}}(\bar{u}; \bar{v}))_\mu \in C_{v_1} \cup O_{v_1})(\bar{x}; \bar{y})
\]

Points 2a and 2b hold by applying the induction hypothesis. Concerning point 2c, notice that if \( D \) is a CB-coderivation then \( O_{v_2} = \emptyset \) by Proposition 58.3. By applying the induction hypothesis on \( v_1 \) and \( v_2 \), we have \( f_{D_{v_2}}(\bar{x}; \bar{y}, \bar{y}) \in B^C \) and \( f_{D_{v_1}}(\bar{x}; \bar{y}) = h(v_1)((\lambda \bar{u} \subseteq \bar{x}, \lambda \bar{v} \subseteq \bar{y}. f_{D_{v_2}}(\bar{u}; \bar{v}))_\mu \in C_{v_1} \cup O_{v_1})(\bar{x}; \bar{y}) \), so that:

\[
f_{D_{v_0}}(\bar{x}; \bar{y}) = h(v_0)((\lambda \bar{u} \subseteq \bar{x}, \lambda \bar{v} \subseteq \bar{y}. f_{D_{v_2}}(\bar{u}; \bar{v}))_\mu \in C_{v_1} \cup O_{v_1})(\bar{x}; \bar{y})
\]

for some \( h(v_0) \).

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

**Corollary 61.** We have the following:

- \( f(\bar{x}) \in C_{v_1} \text{ if and only if } f(\bar{x}) \in \text{FPTIME} \);
- \( f(\bar{x}) \in C_{v_1} \text{ if and only if } f(\bar{x}) \in \text{FELEMENTARY} \).

**Proof sketch.** Soundness (\( \Rightarrow \)) follows from Theorem 55, by showing that \( B^C \) and \( \text{NB}^C \) are closed under simultaneous versions of their recursion schemes. Completeness (\( \Leftarrow \)) follows from Theorem 51 and Theorem 54.

## 7 CONCLUSIONS AND FURTHER REMARKS

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). Along the way we gave novel relativised algebras for these classes based on safe (nested) recursion on well-founded relations.

Since the conference version [11] of this work was published, other works in Cyclic Implicit Complexity have appeared, building on the present work, in particular in the setting of *non-uniform computation* [1, 12].

### 7.1 Unary notation and linear space

It is well-known that \( \text{FLINSPACE} \), i.e. the class of functions computable in linear space, can be captured by reformulating \( B \) in unary notation (see [4]). A similar result can be obtained for \( CB \) by just defining a unary version of the conditional in \( B^C \) (similarly to the ones in [15, 25]) and by adapting the proofs of Lemma 40, Theorem 51 and Theorem 55. On the other hand, \( CNB \) is (unsurprisingly) not sensitive to such choice of notation.
7.2 On unnested recursion with compositions

We believe that Lemma 40 can be adapted to establish a polynomial bound on the growth rate for the function algebra extending $B^-$ by ‘Safe Composition During Safe Recursion’, cf. Remark 35. We conjecture that this function algebra and its extension to recursion on permutation of prefixes (cf. Definition 37) capture precisely the class $\text{FPSPACE}$. Indeed, several function algebras for $\text{FPSPACE}$ have been proposed in the literature, many of which involve variants of (5) (see [28, 32]). These recursion schemes reflect the parallel nature of polynomial space functions, which can be defined in terms of alternating polynomial time computation. We suspect that a circular proof theoretic characterisation of this class can thus be achieved by extending $\text{CB}$ with a ‘parallel’ version of the cut rule and by adapting the left-leaning criterion appropriately. Parallel cuts might also play a fundamental role for potential circular proof theoretic characterisations of circuit complexity classes, like $\text{Alogtime}$ or $\text{NC}$.

7.3 Towards higher-order cyclic implicit complexity

It would be pertinent to pursue higher-order versions of both $\text{CNB}$ and $\text{CB}$, in light of precursory works [15, 25] in circular proof theory as well as ICC [7, 22, 27]. In the case of polynomial-time, for instance, a soundness result for some higher-order version of $\text{CB}$ might follow by translation to (a sequent-style formulation of) Hofmann’s SLR [22]. Analogous translations might be defined for a higher-order version of $\text{CNB}$ once the linearity restrictions on the recursion operator of SLR are dropped. Finally, as SLR is essentially a subsystem of Gödel’s system $\text{T}$, such translations could refine the results on the abstraction complexity (i.e. type level) of the circular version of system $\text{T}$ in [13, 15].

Acknowledgements. We would like to thank Dominik Wehr for pointing out a minor error in the conference version of this paper [11] (see Remark 35). This work was supported by a UKRI Future Leaders Fellowship, ‘Structure vs Invariants in Proofs’ (project reference MR/S035540/1), and by the Wallenberg Academy Fellowship Prolongation project ‘Taming Jörmungandr: The Logical Foundations of Circularity’ (project reference 251080003).

REFERENCES

[1] Matteo Acciaio, Gianluca Curzi, and Giulio Guerrieri. 2024. Infinitary Cut-Elimination via Finite Approximations. In 32nd EACSL Annual Conference on Computer Science Logic, CSL 2024, February 19-23, 2024, Naples, Italy (LIPIcs, Vol. 288). Aniello Murano and Alexandra Silva (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 8:1–8:19. https://doi.org/10.4230/LIPIcs.CSL.2024.8

[2] David Baelde, Amina Doumane, Denis Kuperberg, and Alexis Saurin. 2020. Bouncing threads for infinitary and circular proofs. CoRR abs/2005.08257 (2020). arXiv:2005.08257 https://arxiv.org/abs/2005.08257

[3] David Baelde, Amina Doumane, and Alexis Saurin. 2016. Infinitary Proof Theory: the Multiplicative Additive Case. 62 (2016), 42:1–42:17. https://doi.org/10.4230/LIPIcs.CSL.2016.42

[4] Stephen Bellantoni. 1992. Predicative recursion and computational complexity. PhD thesis.

[5] Stephen Bellantoni. 1995. Predicative Recursion and the Polytyme Hierarchy. In Feasible Mathematics II, Peter Clote and Jeffrey B. Remmel (Eds.). Birkhäuser Boston, Boston, MA, 15–29.

[6] Stephen Bellantoni and Stephen Cook. 1992. A New Recursion-Theoretic Characterization of the Polytyme Functions (Extended Abstract). In Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing (Victoria, British Columbia, Canada) (STOC ’92). Association for Computing Machinery, New York, NY, USA, 283–293. https://doi.org/10.1145/129712.129740

[7] Stephen J. Bellantoni, Karl-Heinz Niggl, and Helmut Schwichtenberg. 2000. Higher type recursion, ramification and polynomial time. Ann. Pure Appl. Log. 104, 1-3 (2000), 37–30. https://doi.org/10.1016/S0168-0072(00)00060-3

[8] Stefano Berardi and Makoto Tatsuta. 2017. Equivalence of inductive definitions and cyclic proofs under arithmetic. In 32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017. IEEE Computer Society, 1–12. https://doi.org/10.1109/LICS.2017.8005114

[9] James Brotherston. 2005. Cyclic Proofs for First-Order Logic with Inductive Definitions. In Automated Reasoning with Analytic Tableaux and Related Methods, International Conference, TABLEAUX 2005, Koblenz, Germany, September 14-17, 2005, Proceedings (Lecture Notes in Computer Science, Vol. 3702), Bernhard Beckert (Ed.). Springer, 78–92. https://doi.org/10.1007/11554554_8
A COMPUTATIONAL EXPRESSIVITY OF (PROGRESSING) CODERIVATIONS

A.1 Examples of coderivations for Proposition 20

In light of Proposition 19, we shall simply omit modalities in (regular) (progressing) coderivations in what follows, i.e. we shall regard any formula in the antecedent of a sequent as modal and we shall omit applications of \(\Box_r\). Consequently, we shall write e.g. cut instead of \(\text{cut}_{2N}\) (and so on) and avoid writing semicolons in the semantics of a coderivation.

From here, Items 1 and 2 from Proposition 20 are proved by way of the following examples.

Example 62 (Extensional completeness at type 1). For any function \(f : \mathbb{N}^k \to \mathbb{N}\) there is a progressing coderivation \(\mathcal{F}\) such that \(f_F = f\). Proceeding by induction on \(k\), if \(k = 0\) then we use the rules 0, \(s_0\), \(s_1\), cut to construct \(\mathcal{F}\) defining the natural number \(f\). Otherwise, suppose \(f : \mathbb{N} \times \mathbb{N}^k \to \mathbb{N}\) and define \(f_0\) as \(f_0(x) = f(n, x)\). We construct the coderivation defining \(f\) as follows:

\[
\begin{array}{c}
\text{cond} \quad \text{N, \text{N} = N} \\
\text{cut} \quad \text{N, \text{N} = N} \\
\end{array}
\]

where \(\mathcal{N}\) is a coderivation converting \(n\) into \(s_1.\ldots.s_10\), and we omit the second premise of \(\text{cond}\) as it is never selected. Notice that \(\mathcal{F}\) is progressing, as red formulas are modal) form a progressing thread contained in the infinite branch that loops on \(\bullet\).

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 \(\mathcal{B}^\infty\)-definable by a regular coderivation. This can be easily inferred from the next two examples using Proposition 19.

Example 63 (Primitive recursion). Let \(f(x, \bar{x})\) be defined by primitive recursion from \(g(\bar{x})\) and \(h(x, \bar{x}, y)\). Given coderivations \(\mathcal{G}\) and \(\mathcal{H}\) defining \(g\) and \(h\) respectively, we construct the following coderivation \(\mathcal{R}\):

\[
\begin{array}{c}
\text{cond} \quad \text{N, \text{N} = N} \\
\text{cut} \quad \text{N, \text{N} = N} \\
\end{array}
\]

where \(\mathcal{N}\) is the coderivation converting \(n\) into \(s_1.\ldots.s_10\), \(\mathcal{P}\) is a coderivation defining unary predecessor, and we avoid writing the second premise of \(\text{cond}\) as it is never selected. Notice that \(\mathcal{R}\) is progressing, as blue formulas \(\mathcal{N}\) (which are modal) form a progressing thread contained in the infinite branch that loops on \(\bullet\). From the associated equational program we obtain:

\[
\begin{align*}
 f_{\mathcal{R}_1}(x, \bar{x}) &= f_{\mathcal{R}_1}(f_{\mathcal{N}}(x), x, \bar{x}) \\
 f_{\mathcal{R}_1}(0, x, \bar{x}) &= g(\bar{x}) \\
 f_{\mathcal{R}_1}(s_1z, x, \bar{x}) &= h(f_{\mathcal{P}}(x), \bar{x}, f_{\mathcal{R}_1}(z, f_{\mathcal{P}}(x), \bar{x}))
\end{align*}
\]

so that \(f_{\mathcal{R}} = f_{\mathcal{R}_1} = f\).

Example 64 (Unbounded search). Let \(g(x, \bar{x})\) be a function, and let \(f(\bar{x}) := \mu x. (g(x, \bar{x}) = 0)\) be the unbounded search function obtained by applying the minimisation operation on \(g(\bar{x})\). Given a coderivation \(\mathcal{G}\) defining \(g\), we construct the
following coderivation $\mathcal{U}$:

\[
\begin{array}{c}
\text{id} \Rightarrow N \Rightarrow N \\
\text{cut} \Rightarrow N \Rightarrow N \\
\text{cut} \Rightarrow N \Rightarrow N \\
\text{cut} \Rightarrow N \Rightarrow N \\
\end{array}
\]

where the coderivation $\mathcal{N}$ computes the unary successor, 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:

\[
\begin{align*}
    f_{\mathcal{U}_0}(x) &= f_{\mathcal{U}_0}(0, x) \\
    f_{\mathcal{U}_1}(x, x) &= f_{\mathcal{U}_1}(f_G(x), x) \\
    f_{\mathcal{U}_1}(0, x, x) &= x \\
    f_{\mathcal{U}_1}(s_0z, x, x) &= f_{\mathcal{U}_1}(f_S(x), x) \quad z \neq 0 \\
    f_{\mathcal{U}_1}(s_1z, x, x) &= f_{\mathcal{U}_1}(f_S(x), x) \\
\end{align*}
\]

Which searches for the least $x \geq 0$ such that $g(x, x) = 0$. Hence, $f_{\mathcal{U}_1}(x) = f(\bar{x})$.

### A.2 Completeness for type 1 primitive recursion

[25] shows that, in the absence of contraction rules, only the primitive recursive functions are so definable (even when using arbitrary 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 [25], 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 [15] (which has explicit contraction).

**Proof of Item 3 from Proposition 20.** First, by Proposition 19 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 [15, Corollary 80].

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

- if $g(\bar{x}) \in T_1(\bar{a})$ and $h(a)(x, \bar{x}) \in T_1(a, \bar{a})$, then the $f(x, \bar{x})$ given by,

  \[
  \begin{align*}
  f(0, \bar{x}) &= g(\bar{x}) \\
  f(s_i x, \bar{x}) &= h_i(\lambda u. f(x, \bar{u}))(x, \bar{x})
  \end{align*}
  \]

is in $T_1(\bar{a})$.

It is not hard to see that the type 1 functions of $T_1$ are precisely those of $T_1(\emptyset)$ (e.g. because of [13, Appendix A]). We now proceed by showing how to define the above scheme by regular and progressing $B^-$-coderivations.
Given a function \( f(\bar{x}) \in T_1(\bar{a}) \), we construct a regular progressing coderivation of \( B^- \),

\[
\begin{align*}
\frac{a_i}{N \Rightarrow N} \quad \frac{D_f(\bar{a})}{\bar{N} \Rightarrow N}
\end{align*}
\]

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

If \( f(\bar{x}) \) is an initial function, an oracle, or \( f(\bar{x}) = h(g(\bar{x}), \bar{x}) \) then the construction is easy (see, e.g., the proof of Theorem 54).

Let us consider the case of recursion (which subsumes usual primitive recursion at type 0). Suppose \( f(x, \bar{x}) \in T_1(\bar{a}) \) where,

\[
\begin{align*}
f(0, \bar{x}) &= g(\bar{x}) \\
1_f (s_0 x, \bar{x}) &= h_0 (\lambda \bar{u}. f(x, \bar{u}))(x, \bar{x}) \\
1_f (s_1 x, \bar{x}) &= h_1 (\lambda \bar{u}. f(x, \bar{u}))(x, \bar{x})
\end{align*}
\]

where \( D_g(\bar{a}) \) and \( D_h(\bar{a}, \bar{a}) \) are already obtained by the inductive hypothesis. We define \( D_f(\bar{a}) \) as follows:

\[
\begin{align*}
\frac{\vdots}{N, \bar{N} \Rightarrow N} & \quad \quad \frac{\vdots}{N, \bar{N} \Rightarrow N} \\
\frac{N, \bar{N} \Rightarrow N}{N, \bar{N} \Rightarrow N} & \quad \quad \frac{\bar{N}, \bar{N} \Rightarrow N \quad N, \bar{N} \Rightarrow N}{N, \bar{N} \Rightarrow N} \\
\frac{\text{cond}}{N, \bar{N} \Rightarrow N}
\end{align*}
\]

Note that the existence of the second and third coderivations above the conditional is given by a similar construction to that of Lemma 53 (see also [15, Lemma 42]).

\[\square\]

### A.3 On the ‘power’ of contraction

Given the computationally equivalent system \( CT_0 \) with contraction from [15], 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 \( B^- + \{c_N, c_{\square N}\} \) the extension of \( B^- \) with the rules \( c_N \) and \( c_{\square N} \) below:

\[
\begin{align*}
\frac{\Gamma, N, N \Rightarrow B}{c_N} & \quad \quad \frac{\Gamma, \square N, \square N \Rightarrow B}{c_{\square N}} \\
\frac{\Gamma, N \Rightarrow B}{c_0}
\end{align*}
\]

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.\(^7\) We have:

**Corollary 65.** \( f(\bar{x};.) \) is definable by a regular progressing \( B^- + \{c_N, c_{\square N}\} \)-coderivation iff it is definable by a regular progressing \( B^- \)-coderivation.

\(^7\)Note that the totality argument of Proposition 16 still applies in the presence of these rules, cf. also [15].
Proof idea. The former system is equivalent to $\text{CT}_0$ from [15], whose type 1 functions are just those of $\text{T}_1$ by [15, Corollary 80], which are all defined by regular progressing $\mathbb{B}^-$-coderivations by Proposition 20 Item 3.

Remark 66. The reader may at this point wonder if a direct ‘contraction-admissibility’ argument exists for the rules in (26). First, notice that $c_N$ can be derived in $\mathbb{B}^-$:

$$
\begin{align*}
\text{id} & : N \Rightarrow N \\
\text{w} & : \Gamma, N \Rightarrow N \quad \Gamma, N, N \Rightarrow B \\
\text{cut} & : N, N \Rightarrow N \quad \Gamma, N \Rightarrow B
\end{align*}
$$

While a similar derivation exists for $c_{\mathbb{C}_2}$, note that this crucially does not preserve the same notion of thread (cf. colours above) and so does not, a priori, preserve progressiveness.

Example 67 (Ackermann-Péter). As suggested by Proposition 20 Item 3, regular progressing $\mathbb{B}^-$-coderivations are able to define the Ackermann-Péter function $A(x, y)$,

$$
A(0, y) = x + 1 \\
A(x + 1, 0) = A(x, 1) \\
A(x + 1, y + 1) = A(x, A(x + 1, y))
$$

which is well-known to not be (type 0) primitive recursive, but is nevertheless type 1 primitive recursive.

The usual cyclic representation of $A(x, y)$ using only base types, e.g. as in [15, 25], mimics the equational program above, namely applying case analysis on the first input before applying case analysis on the second input. It is important in this construction to be able to explicitly contract the first input, corresponding to $x$, due to the third line of the equational program above.

In our context-sharing setting, without explicit contraction, we may nonetheless represent $A(x, y)$ by conducting a case analysis on the second input $y$ first, and including some redundancy, as in the regular progressing coderivation in Figure 6. To facilitate readability, we again omit modalities and work purely in unary notation, representing a natural number $n$ by $s_1 \cdot s_1 \cdot \ldots \cdot s_1(0)$.

The verification of the semantics of this coderivation is routine. To see that it is progressing, we may conduct a case analysis on the set $L$ of infinitely often visited loops, among (1), (2), (3), in an arbitrary branch:

- if $L = \emptyset$ then the branch is finite;
- if $L = \{(1)\}$ then there is a progressing thread on the red $N$;
• if \( L = \{(2)\} \) then there is a progressing thread on the blue \( N \) (note that the red \( N \) thread does not progress);
• if \( L = \{(3)\} \) then there is a progressing thread on the red \( N \);
• if \( L = \{(1), (2)\} \) then there is a progressing thread on the red \( N \): on each iteration of (1) the thread progresses, and remains intact (but does not progress) on each iteration of (2);
• if \( L = \{(1), (3)\} \) then there is a progressing thread on the red \( N \): the thread progresses on each iteration of either (1) or (3);
• if \( L = \{(2), (3)\} \) then there is a progressing thread on the red \( N \): on each iteration of (2) the thread remains intact (but does not progress), and on each iteration of (3) the thread progresses;
• if \( L = \{(1), (2), (3)\} \) then there is a progressing thread on the red \( N \): the thread progresses on each iteration of (1) or (3), and remains intact (but does not progress) on each iteration of (3).

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009