Non-Uniform Complexity via Non-Wellfounded Proofs

Gianluca Curzi
University of Birmingham, UK

Anupam Das
University of Birmingham, UK

Abstract

Cyclic and non-wellfounded proofs are now increasingly employed to establish metalogical results in a variety of settings, in particular for type systems with forms of (co)induction. Under the Curry-Howard correspondence, a cyclic proof can be seen as a typing derivation "with loops", closer to low-level machine models, and so comprise a highly expressive computational model that nonetheless enjoys excellent metalogical properties.

In recent work, we showed how the cyclic proof setting can be further employed to model computational complexity, yielding characterisations of the polynomial time and elementary computable functions. These characterisations are "implicit", inspired by Bellantoni and Cook's famous algebra of safe recursion, but exhibit greater expressivity thanks to the looping capacity of cyclic proofs.

In this work we investigate the capacity for *non-wellfounded* proofs, where finite presentability is relaxed, to model non-uniformity in complexity theory. In particular, we present a characterisation of the class **FP**/*poly* of functions computed by polynomial-size circuits. While relating non-wellfoundedness to non-uniformity is a natural idea, the precise amount of irregularity, informally speaking, required to capture **FP**/*poly* is given by proof-level conditions novel to cyclic proof theory. Along the way, we formalise some (presumably) folklore techniques for characterising non-uniform classes in relativised function algebras with appropriate oracles.

2012 ACM Subject Classification Theory of computation → Complexity theory and logic; Theory of computation → Proof theory

Keywords and phrases Cyclic proofs, non-wellfounded proof-theory, non-uniform complexity, polynomial time, safe recursion, implicit complexity

Digital Object Identifier 10.4230/LIPIcs.CSL.2023.16

Related Version *Full Version*: `https://arxiv.org/abs/2211.16104`

Funding Both authors are supported by a UKRI Future Leaders Fellowship, *Structure vs. Invariants in Proofs*, project reference MR/S035540/1.

Acknowledgements We thank the anonymous reviewers for their helpful comments and suggestions.

1 Introduction

Non-wellfounded proof theory is the study of possibly infinite (but finitely branching) proofs, where appropriate global correctness criteria guarantee logical consistency. This area originates (in its modern guise) in the context of the modal μ-calculus [31, 16], serving as an alternative framework to manipulate least and greatest fixed points, and hence to model inductive and coinductive reasoning. Since then, non-wellfounded proofs have been widely investigated in many respects, such as predicate logic [8, 6], algebras [14, 15], arithmetic [32, 5, 12], proofs-as-programs interpretations [2, 17, 11, 24, 13], and continuous cut-elimination [30, 18]. Special attention in these works is drawn to *cyclic* (or *regular*) proofs, i.e. non-wellfounded proofs with only finitely many distinct subproofs, comprising a natural notion of finite presentability in terms of (possibly cyclic) directed graphs.

The *Curry-Howard* reading of non-wellfounded proofs has revealed a deep connection between proof-theoretic properties and computational behaviours [11, 24, 13]. On the one hand, the typical correctness conditions ensuring consistency, called *progressing* (or *validity*) criteria, correspond to totality: functions computed by progressing proofs are always well-defined on all arguments. On the other hand, regularity has a natural counterpart in the notion of *uniformity*: circular proofs can be properly regarded as programs, i.e. as finite sets of machine instructions, thus having a "computable" behaviour.

In a recent work [10], the authors extended these connections between non-wellfounded proof theory and computation to the realm of *computational complexity.* We introduced the proof systems CB and CNB capturing, respectively, the class of functions computable in polynomial time (**FP**) and the elementary functions (**FELEMENTARY**). These proof systems are defined by identifying global conditions on circular progressing proofs motivated by ideas from *Implicit Computational Complexity* (ICC).

ICC, broadly construed, is the study of machine-free (and often bound-free) characterisations of complexity classes. One of the seminal works in the area is Bellantoni and Cook's function algebra B for **FP** based on *safe recursion* [4]. The prevailing idea behind safe recursion (and its predecessor, *ramified recursion* [26]) is to partition function arguments into "safe" and "normal", namely writing $f(x_1, \ldots, x_m; y_1, \ldots, y_n)$ when f takes m normal inputs $\vec{x}$ and n safe inputs $\vec{y}$. In functions of B, the recursive parameters are always normal arguments, while recursive calls can only appear in safe position; hence, no recursive call can be used as recursive parameters of other previously defined functions. Our system CB morally represents a cyclic proof theoretic formulation of B.

To establish the characterisation result for CB we developed a novel function algebra for **FP**, called $\mathsf{B}^{\subset}$. Roughly, the latter extends B with a more expressive recursion mechanism on a special well-founded preorder, "$\subset$", based on permutation of prefixes of normal arguments, and whose definition requires relativisation of the algebra to admit *oracles.* The characterisation theorem is then obtained by a "sandwich" technique, where the function algebras B and $\mathsf{B}^{\subset}$ serve, respectively, as lower and upper bounds for CB.

In this paper we investigate the computational interpretation of more general *non-wellfounded* proofs, where finite presentability is relaxed in order to model *non-uniform complexity.* In particular we consider the class **FP**/*poly* of functions computable in polynomial time by Turing machines with access to *polynomial advice.* Equivalently, **FP**/*poly* is the class of functions computed by families of polynomial-size circuits. Note, in particular, that **FP**/*poly* includes *undecidable problems*, and so cannot be characterised by purely cyclic proof systems or usual function algebras, which typically have only computable functions.

We define the system nuB ("non-uniform B"), allowing a form of non-wellfoundedness somewhere between arbitrary non-wellfounded proofs and full regularity, and show that nuB duly characterises **FP**/*poly.* The characterisation theorem for nuB relies on an adaption of the aforementioned sandwich technique for CB to the current setting. This requires a relativisation of both B and $\mathsf{B}^{\subset}$ to a set of oracles, which we call $\mathbb{R}$, deciding properties of string length. As a byproduct of our proof method we also obtain new relativised function algebras for **FP**/*poly* based on safe recursion, $\mathsf{B}(\mathbb{R})$ and $\mathsf{B}^{\subset}(\mathbb{R})$; these are folklore-style results that, as far as we know, have not yet appeared in the literature.

The overall structure of our result relies on a "grand tour" of inclusions, summarised as:

$$\mathbf{FP}/\mathit{poly} \overset{\text{P.27}}{\subseteq} \mathsf{B}(\mathbb{R}_{1;0}) \overset{\text{P.32}}{\subseteq} \mathsf{CB}(\mathbb{R}_{1;0}) \overset{\text{P.33}}{\subseteq} \mathsf{nuB} \overset{\text{T.36}}{\subseteq} \mathsf{CB}(\mathbb{R}_{1;1}) \overset{\text{L.43}}{\subseteq} \mathsf{B}^{\subset}(\mathbb{R}_{1;1}) \overset{\text{P.42}}{\subseteq} \mathbf{FP}(\mathbb{R}) \overset{\text{P.26}}{\subseteq} \mathbf{FP}/\mathit{poly}$$

While this may seem like a long route to take, the structure of our argument is designed so that each of the above inclusions are relatively simple to establish and, as we said, yields several intermediate characterisions of **FP**/*poly* of self-contained interest.

Related work. Characterisations of non-uniform complexity classes in the style of ICC have been considered in the context of the λ-calculus [27] and variants of linear logic [29]. The former captures the class $\mathbf{P}/\mathit{poly}$, i.e., the languages decided by families of polynomial circuits, while the latter also captures $\mathbf{L}/\mathit{poly}$, i.e., the languages decided by families of polynomial size branching programs (i.e. decision trees with sharing). However this is the first work (as far as we know) that attempts to relate non-wellfoundedness in proof theory to non-uniformity in complexity theory.

The relativised proof systems and function algebras presented in this paper only query "bits of real numbers". Proof systems based on linear logic and implementing polytime computation over actual binary streams have been considered, e.g., in [20], which provide an ICC-like characterisation of Ko's class of polynomial time computable functions over real numbers [23].

Outline of the paper. This paper is structured as follows. In Section 2 we recall some preliminaries on non-uniform and implicit complexity, in particular a proof theoretic formulation of the algebra B. In Section 3 we recall the circular system CB from [10], and introduce our new system nuB. In Section 4 we take an interlude to present some *relativised* characterisations of $\mathbf{FP}/\mathit{poly}$, both in the machine setting and the implicit setting, that will later serve use in our grand tour of inclusions. In Section 5 we employ those characterisations to establish the lower bound for nuB, and in Section 6 we recast nuB as a sort of relativised circular system. Finally in Section 7 we adapt results from [10] translating circular proofs to an appropriate function algebra to the relativised setting, thereby achieving the upper bound for nuB.

2 Preliminaries on computational complexity and safe recursion

Throughout this work we only consider (partial) functions on *natural numbers*. We write $|x|$ for the length of the binary representation of a number x, and for lists of arguments $\vec{x} = x_1, \dots, x_n$ we write $|\vec{x}|$ for the list $|x_1|, \dots, |x_n|$.

2.1 Non-uniform complexity classes

$\mathbf{FP}$ is the class of (total) functions computable in polynomial time on a Turing machine. The "non-uniform" class $\mathbf{FP}/\mathit{poly}$ is an extension of $\mathbf{FP}$ that intuitively has access to a polynomial amount of "advice", determined only by the *length* of the input. Formally:

▶ **Definition 1** (Non-uniform polynomial time). $\mathbf{FP}/\mathit{poly}$ is the class of functions $f(\vec{x})$ for which there are strings $\alpha_{\vec{n}} \in \{0,1\}^*$, of size polynomial in $\vec{n}$, and some $f'(x, \vec{x}) \in \mathbf{FP}$ with:

- $|\alpha_{\vec{n}}|$ is polynomial in $\vec{n}$.
- $f(\vec{x}) = f'(\alpha_{|\vec{x}|}, \vec{x})$.

The strings $\{\alpha_{\vec{n}}\}_{\vec{n}}$ represent the *polynomial advice* given to a polynomial-time computation, here $f'(x, \vec{x})$. Note that $f(\vec{x})$ only "receives advice' depending on the lengths of its inputs, $\vec{x}$.

Note, in particular, that $\mathbf{FP}/\mathit{poly}$ admits undecidable problems. E.g. the function $f(x) = 1$ just if $|x|$ is the code of a halting Turing machine (and 0 otherwise) is in $\mathbf{FP}/\mathit{poly}$. Indeed, the point of the class $\mathbf{FP}/\mathit{poly}$ is to rather characterise a more non-uniform notion of computation. In particular, the following is well-known (see, e.g., [1, Theorem 6.11]):

▶ **Proposition 2.** *$f(\vec{x}) \in \mathbf{FP}/\mathit{poly}$ iff there are polynomial-size circuits computing $f(\vec{x})$.*

2.2 The Bellantoni-Cook algebra

A *two-sorted* function is a function $f(\vec{x};\vec{y})$ whose arguments have been delimited into "normal" ones ($\vec{x}$, left of ";"), and "safe" ones ($\vec{y}$, right of ";").

The two-sorted algebra B was introduced in [4] and is defined as follows:

▶ **Definition 3** (Bellantoni-Cook)**.** B is the smallest class of two-sorted functions containing,

- $0(;) := 0$
- $\mathsf{s}_0(;x) := 2x$
- $\mathsf{s}_1(;x) := 2x+1$
- $\pi^{m;n}_{j;}(x_0,\dots,x_{m-1};y_0,\dots,y_{n-1}) := x_j$, whenever $j<m$.
- $\pi^{m;n}_{;j}(x_0,\dots,x_{m-1};y_0,\dots,y_{n-1}) := y_j$, whenever $j<n$.
- $\mathsf{p}(;x) = \lfloor \frac{x}{2} \rfloor$
- $\mathsf{cond}(;w,x,y,z) := \begin{cases} x & w=0 \\ y & w=0 \mod 2, w\neq 0 \\ z & w=1 \mod 2 \end{cases}$

and closed under:

- (Safe composition)
  - if $g(\vec{x};) \in \mathsf{B}$ and $h(\vec{x},x;\vec{y}) \in \mathsf{B}$ then also $f(\vec{x};\vec{y}) \in \mathsf{B}$ where $f(\vec{x};\vec{y}) := h(\vec{x},g(\vec{x};);\vec{y})$.
  - if $g(\vec{x};\vec{y}) \in \mathsf{B}$ and $h(\vec{x};\vec{y},y) \in \mathsf{B}$ then also $f(\vec{x};\vec{y}) \in \mathsf{B}$ where $f(\vec{x};\vec{y}) := h(\vec{x};\vec{y},g(\vec{x};\vec{y}))$
- (Safe recursion on notation) if $g(\vec{x};\vec{y}) \in \mathsf{B}$ and $h_0(x,\vec{x};\vec{y},y), h_1(x,\vec{x};\vec{y},y) \in \mathsf{B}$ then also $f(x,\vec{x};\vec{y}) \in \mathsf{B}$ where:

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

Safe composition ensures that safe arguments may never appear in a normal position. Note that, in the recursion scheme, the recursion parameter is always a normal argument, whereas recursive calls must appear in safe position. Along with the constraints on safe composition, this ensures that the position of a recursive call is never the recursion parameter of another recursion. This seemingly modest constraint duly restricts computation to polynomial time, yielding Bellantoni and Cook's main result:

▶ **Theorem 4** ([4])**.** *$f(\vec{x}) \in \mathbf{FP}$ if and only if $f(\vec{x};) \in \mathsf{B}$.*

2.3 A proof-theoretic presentation of Bellantoni-Cook

We shall work with a formulation of B as a $S4$-style type system in sequent-calculus style, where modalities are used to distinguish the two sorts (similarly to [21]).

We consider *types* (or *formulas*) N ("safe") and $\Box N$ ("normal") which intuitively vary over the natural numbers. We write A, B, etc. to vary over types. A *sequent* is an expression $\Gamma \Rightarrow A$, where Γ is a list of types (called the *context* or *antecedent*) and A is a type (called the *succedent*). For a list of types $\Gamma = \overbrace{N,\dots,N}^{k}$, we write $\Box\Gamma$ for $\overbrace{\Box N,\dots,\Box N}^{k}$.

▶ **Definition 5.** A B-*derivation* is a (finite) derivation built from the rules in Figure 1.

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

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

$$0\ \frac{}{\Rightarrow N} \qquad 1\ \frac{}{\Rightarrow N} \qquad \mathsf{s}_0\ \frac{\Gamma \Rightarrow A}{\Gamma \Rightarrow A} \qquad \mathsf{s}_1\ \frac{\Gamma \Rightarrow A}{\Gamma \Rightarrow A} \qquad \mathsf{srec}\ \frac{\Gamma \Rightarrow N \quad \Box N, \Gamma, N \Rightarrow N \quad \Box N, \Gamma, N \Rightarrow N}{\Box N, \Gamma \Rightarrow N}$$

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

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

Figure 1 B as a sequent-style type system.

$$\begin{aligned}
f_{\mathsf{id}}(;y) &:= y \\
f_{\mathsf{cut}_N(\mathcal{D}_0,\mathcal{D}_1)}(\vec{x};\vec{y}) &:= f_{\mathcal{D}_1}(\vec{x};\vec{y}, f_{\mathcal{D}_0}(\vec{x};\vec{y})) \\
f_{\mathsf{cut}_\Box(\mathcal{D}_0,\mathcal{D}_1)}(\vec{x};\vec{y}) &:= f_{\mathcal{D}_1}(f_{\mathcal{D}_0}(\vec{x};\vec{y}), \vec{x};\vec{y}) \\
f_{\mathsf{w}_N(\mathcal{D}_0)}(\vec{x};\vec{y},y) &:= f_{\mathcal{D}_0}(\vec{x};\vec{y}) \\
f_{\mathsf{w}_\Box(\mathcal{D}_0)}(x,\vec{x};\vec{y}) &:= f_{\mathcal{D}_0}(\vec{x};\vec{y}) \\
f_{\mathsf{e}_N(\mathcal{D}_0)}(\vec{x};\vec{y},y,y',\vec{y}') &:= f_{\mathcal{D}_0}(\vec{x};\vec{y},y',y,\vec{y}') \\
f_{\mathsf{e}_\Box(\mathcal{D}_0)}(\vec{x},x,x',\vec{x}';\vec{y}) &:= f_{\mathcal{D}_0}(\vec{x},x',x,\vec{x}';\vec{y}) \\
f_{\Box_l(\mathcal{D}_0)}(x,\vec{x};\vec{y}) &:= f_{\mathcal{D}_0}(\vec{x};\vec{y},x) \\
f_{\Box_r(\mathcal{D}_0)}(\vec{x};) &:= f_{\mathcal{D}_0}(\vec{x};) \\
f_i(;) &:= i \\
f_{\mathsf{s}_i(\mathcal{D}_0)}(\vec{x};\vec{y}) &:= \mathsf{s}_i(; f_{\mathcal{D}_0}(\vec{x};\vec{y}))
\end{aligned}
\qquad
\begin{aligned}
f_{\mathsf{srec}(\mathcal{D}_0,\mathcal{D}_1,\mathcal{D}_2)}(0,\vec{x};\vec{y}) &:= f_{\mathcal{D}_0}(\vec{x};\vec{y}) \\
f_{\mathsf{srec}(\mathcal{D}_0,\mathcal{D}_1,\mathcal{D}_2)}(\mathsf{s}_i x,\vec{x};\vec{y}) &:= f_{\mathcal{D}_{i+1}}(x,\vec{x};\vec{y}, \\
&\qquad f_{\mathsf{srec}(\mathcal{D}_0,\mathcal{D}_1,\mathcal{D}_2)}(x,\vec{x};\vec{y})) \\
f_{\mathsf{cond}_N(\mathcal{D}_0,\mathcal{D}_1,\mathcal{D}_2)}(\vec{x};\vec{y},0) &:= f_{\mathcal{D}_0}(\vec{x};\vec{y}) \\
f_{\mathsf{cond}_N(\mathcal{D}_0,\mathcal{D}_1,\mathcal{D}_2)}(\vec{x};\vec{y},\mathsf{s}_i y) &:= f_{\mathcal{D}_{i+1}}(\vec{x};\vec{y},y) \\
f_{\mathsf{cond}_\Box(\mathcal{D}_0,\mathcal{D}_1,\mathcal{D}_2)}(0,\vec{x};\vec{y}) &:= f_{\mathcal{D}_0}(\vec{x};\vec{y}) \\
f_{\mathsf{cond}_\Box(\mathcal{D}_0,\mathcal{D}_1,\mathcal{D}_2)}(\mathsf{s}_i x,\vec{x};\vec{y}) &:= f_{\mathcal{D}_{i+1}}(x,\vec{x};\vec{y}) \\
f_{|\mathsf{cond}|_N(\mathcal{D}_0,\mathcal{D}_1)}(\vec{x};\vec{y},0) &:= f_{\mathcal{D}_0}(\vec{x};\vec{y}) \\
f_{|\mathsf{cond}|_N(\mathcal{D}_0,\mathcal{D}_1)}(\vec{x};\vec{y},\mathsf{s}_i y) &:= f_{\mathcal{D}_1}(\vec{x};\vec{y},y) \\
f_{|\mathsf{cond}|_\Box(\mathcal{D}_0,\mathcal{D}_1)}(0,\vec{x};\vec{y}) &:= f_{\mathcal{D}_0}(\vec{x};\vec{y}) \\
f_{|\mathsf{cond}|_\Box(\mathcal{D}_0,\mathcal{D}_1)}(\mathsf{s}_i x,\vec{x};\vec{y}) &:= f_{\mathcal{D}_1}(x,\vec{x};\vec{y})
\end{aligned}$$

Figure 2 Semantics of system B, where $i \in \{0,1\}$ and $\mathsf{s}_i x \neq 0$ and $\mathsf{s}_i y \neq 0$.

The colouring of type occurrences in Figure 1 may be ignored for now, they will become relevant in the next section. We may write $\mathcal{D} = \mathsf{r}(\mathcal{D}_1, \dots, \mathcal{D}_n)$ (for $n \leq 3$) if r is the bottom-most inference step of a derivation $\mathcal{D}$ whose immediate subderivations are, respectively, $\mathcal{D}_1, \dots, \mathcal{D}_n$. As done in [10], we shall assume w.l.o.g. that sequents have shape $\Box N, \dots, \Box N, N, \dots, N \Rightarrow A$, i.e. in the left-hand side all $\Box N$ occurrences are placed before all N occurrences.

We construe the system of B-derivations as a class of two-sorted functions by identifying each rule instance as an operation on two-sorted functions as follows:

▶ **Definition 6** (Semantics of B). Given a B-derivation $\mathcal{D}$ of $\overbrace{\Box N, \dots, \Box N}^{m}, \overbrace{N, \dots, N}^{n} \Rightarrow A$ we define a two-sorted function $f_{\mathcal{D}}(x_1, \dots, x_m; y_1, \dots, y_n)$ in Figure 2 by induction on the structure of $\mathcal{D}$ (all rules as typeset in Figure 1).

This formal semantics exposes how B-derivations and functions in the algebra B relate. The rule srec in Figure 1 corresponds to safe recursion, and safe composition along safe parameters is expressed by cut_N. Note, however, that the interpretation of $\mathsf{cut}_\Box$ in Figure 2 apparently does not satisfsy the required constraint on safe composition of a function g along a normal parameter of a function h, which forbids the presence of safe parameters in g. However, this admission turns out to be harmless, and we are able to obtain the following result that justifies the overloading of the notation "B":

▶ **Proposition 7** ([10]). *$f(\vec{x};\vec{y}) \in \mathsf{B}$ iff there is a B-derivation $\mathcal{D}$ for which $f_{\mathcal{D}}(\vec{x};\vec{y}) = f(\vec{x};\vec{y})$.*

▶ Remark 8 (Bootstrapping). Note that the rules 1, $|\mathsf{cond}|_N$ and $|\mathsf{cond}|_\Box$ are semantically redundant, being derivable from the others by: $f_1 = f_{\mathsf{s}_1(0)}$, $f_{|\mathsf{cond}|_N(\mathcal{D}_0,\mathcal{D}_1)} = f_{\mathsf{cond}_N(\mathcal{D}_0,\mathcal{D}_1,\mathcal{D}_1)}$, and $f_{|\mathsf{cond}|_\Box(\mathcal{D}_0,\mathcal{D}_1)} = f_{\mathsf{cond}_\Box(\mathcal{D}_0,\mathcal{D}_1,\mathcal{D}_1)}$. Indeed, our original presentation of the system in [10] did not include these rules, but we have "bootstrapped" our system here in order to facilitate the definitions of our restricted "non-wellfounded" systems later for characterising **FP**/*poly*, in particular in Section 3.

3 Non-wellfounded systems based on Bellantoni-Cook

In this section we recall a "coinductive" version of B that was recently introduced in our earlier work [10], and go on to introduce the new system nuB of this work. In particular we shall give global criteria that control the computational strength of non-wellfounded typing derivations. Throughout this section we shall work with the system $\mathsf{B}^- := \mathsf{B} - \{\mathsf{srec}\}$.

▶ **Definition 9** (Coderivations). A (B^--)*coderivation* $\mathcal{D}$ is a possibly infinite rooted tree generated by the rules of B^-. Formally, we identify $\mathcal{D}$ with a (labelled) prefix-closed subset of $\{0,1,2\}^*$ (i.e. a ternary tree). Each node is labelled by an inference step from B^- such that, whenever $\nu \in \mathcal{D}$ is labelled by a step $\dfrac{S_1 \quad \cdots \quad S_n}{S}$, for $n \leq 3$, ν has n children in $\mathcal{D}$ labelled by steps with conclusions $S_1, \dots, S_n$ respectively. Sub-coderivations of a coderivation $\mathcal{D}$ rooted at position $\nu \in \{0,1,2\}^*$ are denoted $\mathcal{D}_\nu$, so that $\mathcal{D}_\varepsilon = \mathcal{D}$.

Examples of coderivations can be found in Figure 3 (some of them are from [10]), whose computational meaning is discussed in Example 13, and employ the following conventions:

▶ **Convention 10** (Representing coderivations). Henceforth, we may mark steps by • (or similar) in a 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 structural rules e_N, $\mathsf{e}_\Box$, w_N and $\mathsf{w}_\Box$, absorbing them into other steps (typically cuts) when it causes no confusion. 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, we may compress them into a single "parametrised" sub-coderivation $\mathcal{D}_i$ (with $i = 0, 1$).

As discussed in [13, 11, 24], coderivations can be identified with Kleene-Herbrand-Gödel style equational programs, in general computing partial recursive functionals (see, e.g., [22, §63] for further details). We shall specialise this idea to our two-sorted setting.

▶ **Definition 11** (Semantics of coderivations). To each B^--coderivation $\mathcal{D}$ we associate a two-sorted Kleene-Herbrand-Gödel partial function $f_{\mathcal{D}}$ obtained by construing the semantics of Definition 6 as a (possibly infinite) equational program. Given a two-sorted function $f(\vec{x};\vec{y})$, we say that f is *defined* by a B^--coderivation $\mathcal{D}$ if $f_{\mathcal{D}}(\vec{x};\vec{y}) = f(\vec{x};\vec{y})$.

▶ Remark 12. The notion of *computation* for equational programs is given by (finitary) reasoning in equational logic (see, e.g., [22, §63]): for numerals $\vec{m}, \vec{n}$, we have that $f_{\mathcal{D}}(\vec{m};\vec{n})$ is well-defined and returns some numeral k just if the equation $f_{\mathcal{D}}(\vec{m};\vec{n}) = k$ can be (finitely)

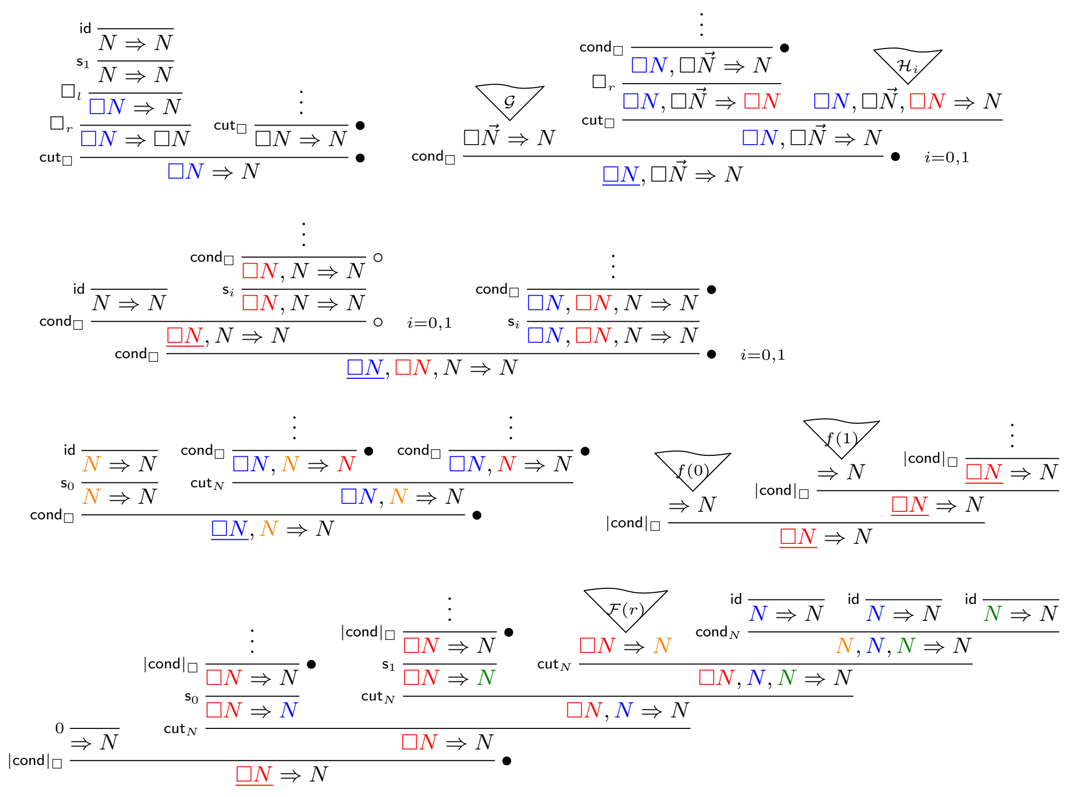


Figure 3 Examples of coderivations: $\mathcal{I}$ (top left), $\mathcal{R}$ (top right), $\mathcal{C}$ (second line), $\mathcal{E}$ (third line, left), $\mathcal{F}(f)$ with $f : \mathbb{N} \to \mathbb{N}$ (third line, right), $\mathcal{A}(r)$ with $r : \mathbb{N} \to \{0, 1\}$ (bottom).

derived in equational logic (with basic numerical axioms) over the equational program for $\mathcal{D}$. Implicit here is the fact that the semantics of B^--coderivations yield *coherent* equational programs: whenever $f_{\mathcal{D}}(\vec{m};\vec{n}) = k$ and $f_{\mathcal{D}}(\vec{m};\vec{n}) = k'$ are derivable then $k = k'$ [13, 11].

▶ **Example 13.** By purely equational reasoning, we can simplify the Kleene-Gödel-Herbrand style semantics in Definition 11 of the coderivations in Figure 3 to get the equational programs in Figure 4: $f_{\mathcal{I}}$ represents a function that is always undefined, as its equational program keeps increasing the length of the input; $f_{\mathcal{R}}$ is an instance of a *non-safe* recursion scheme (on notation), as the recursive call appears in normal position; $f_{\mathcal{C}}$ computes concatenation of the binary representation of three natural numbers; $f_{\mathcal{E}}$ has exponential growth rate (as long as $y \neq 0$), since $f_{\mathcal{E}}(x; y) = 2^{2^{|x|}} \cdot |y|$; the (infinite) equational program for $f_{\mathcal{F}(f)}$ computes $f(|x|)$ by simply exhausting the values of $|x|$; finally, $f_{\mathcal{A}(r)}$ on input x returns the binary string $r(0) \cdot r(1) \cdot \cdots \cdot r(|x| - 1)$ if $x > 0$, and 0 otherwise.

The above examples illustrate several recursion theoretic features of B^--coderivations that we shall seek to control in the remainder of this section:

(I) *non-totality* (e.g., the coderivation $\mathcal{I}$);

(II) *non-computability* (e.g., the coderivation $\mathcal{F}(f)$, with f non-computable);

(III) *non-safety* (e.g., the coderivation $\mathcal{R}$), despite the presence of modalities implementing the normal/safe distinction of function arguments;

(IV) *nested recursion* (e.g., the coderivation $\mathcal{E}$).

$$
\begin{aligned}
f_{\mathcal{I}}(x;) &= f_{\mathcal{I}}(\mathsf{s}_1 x;) \\
f_{\mathcal{R}}(0, \vec{x};) &= f_{\mathcal{G}}(\vec{x};) \\
f_{\mathcal{R}}(\mathsf{s}_i x, \vec{x};) &= f_{\mathcal{H}_i}(x, \vec{x}, f_{\mathcal{R}}(x, \vec{x};);) \\
f_{\mathcal{C}}(0, 0; z) &= z \\
f_{\mathcal{C}}(0, \mathsf{s}_i y; z) &= \mathsf{s}_i f_{\mathcal{C}}(0, y; z) \\
f_{\mathcal{C}}(\mathsf{s}_i x, y; z) &= \mathsf{s}_i f_{\mathcal{C}}(x, y; z)
\end{aligned}
\qquad
\begin{aligned}
f_{\mathcal{E}}(0; y) &= \mathsf{s}_0(; y) \\
f_{\mathcal{E}}(\mathsf{s}_i x; y) &= f_{\mathcal{E}}(x; f_{\mathcal{E}}(x; y)) \\
\{f_{\mathcal{F}(f)}(x;) &= f(|x|)\}_{|x| \in \mathbb{N}} \\
f_{\mathcal{A}(r)}(0;) &= 0 \\
f_{\mathcal{A}(r)}(\mathsf{s}_i x;) &= \begin{cases} \mathsf{s}_0 f_{\mathcal{A}(r)}(x;) & \text{if } f_{\mathcal{F}(r)}(x;) = 0 \\ \mathsf{s}_1 f_{\mathcal{A}(r)}(x;) & \text{otherwise} \end{cases}
\end{aligned}
$$

Figure 4 Equational programs derived from the coderivations in Figure 3, where $i \in \{0, 1\}$.

To address (I) we shall adapt to our setting a well-known "totality criterion" from non-wellfounded proof theory (similar to those in [13, 11, 24]). First we need to recall some standard structural proof theoretic notions:

▶ **Definition 14** (Ancestry). Fix a coderivation $\mathcal{D}$. We say that a type occurrence A is an *immediate ancestor* of a type occurrence B in $\mathcal{D}$ if they are types in a premiss and conclusion (respectively) of an inference step and, as typeset in Figure 1, have the same colour. If A and B are in some Γ or Γ', then furthermore they must be in the same position in the list.

For a definition of immediate ancestry avoiding colours, we point the reader to standard proof theory references, e.g. [9, Sec. 1.2.3]. Being a binary relation, immediate ancestry forms a directed graph upon which our totality criterion is built:

▶ **Definition 15** (Progressing coderivations). Fix a coderivation $\mathcal{D}$. A *thread* is a maximal path in the graph of immediate ancestry. We say that a (infinite) thread is *progressing* if it is eventually constant $\Box N$ and infinitely often principal for a $\mathsf{cond}_\Box$ rule or a $|\mathsf{cond}|_\Box$ rule. A coderivation is *progressing* if each of its infinite branches has a progressing thread.

In [10] we showed that the progressing criterion is indeed sufficient (but obviously not necessary) to guarantee that the partial function computed by a coderivation is, in fact, total (see also [24, 11, 13]):

▶ **Proposition 16** (Progressing implies totality, [10]). *If $\mathcal{D}$ is progressing, then $f_{\mathcal{D}}$ is total.*

The argument for this proposition is by contradiction: assuming non-totality, construct an infinite "non-total branch", whence a contradiction to well-orderedness of $\mathbb{N}$ is implied by a progressing thread along it. We shall use similar argument later in the proof of Lemma 37.

▶ **Example 17.** In Figure 3, $\mathcal{I}$ has precisely one infinite branch (that loops on $\bullet$) which contains no instances of $\mathsf{cond}_\Box$ or $|\mathsf{cond}|_\Box$ at all, so $\mathcal{I}$ is not progressing. On the other hand, $\mathcal{C}$ has two simple loops, one on $\bullet$ and the other one on $\circ$. For any infinite branch B we have two cases: if B crosses the bottommost conditional infinitely many times, it contains a progressing blue thread; otherwise, B crosses the topmost conditional infinitely many times, so that it contains a progressing red thread. Therefore, $\mathcal{C}$ is progressing. By applying the same reasoning, we conclude that $\mathcal{E}$, $\mathcal{F}(f)$, $\mathcal{A}(r)$, and $\mathcal{R}$ are progressing (if $\mathcal{G}$ and $\mathcal{H}_i$ are).

To address (III)-(IV) we recall the following properties of coderivations from [10]:

▶ **Definition 18** (Safety, left-leaning). We say that a coderivation $\mathcal{D}$ is *safe* if each branch crosses only finitely many $\mathsf{cut}_\Box$-steps, and *left-leaning* if each branch goes right at a cut_N-step only finitely often.

▶ **Example 19.** In Figure 3, the only non-safe coderivations are $\mathcal{R}$ and $\mathcal{I}$, as the branches looping on $\bullet$ contain infinitely many $\mathsf{cut}_\Box$. $\mathcal{E}$ is the only non-left-leaning coderivation, as it has a branch looping at $\bullet$ that crosses infinitely many times the rightmost premise of a cut_N.

Finally, concerning (II), recall that the aim of this work is to characterise non-uniform classes, which may contain non-computable predicates and functions. To this end we introduce a generalisation of the notion of "regularity", typically corresponding to computability (e.g. in [11, 13, 24]), that is commonplace in cyclic proof theory:

▶ **Definition 20** (Generalised regularity)**.** Let $\mathsf{R} \subseteq \mathsf{B}^-$. A B^--coderivation $\mathcal{D}$ is R-*regular* if it has only finitely many distinct sub-coderivations containing rules among R. If $\mathsf{R} = \mathsf{B}^-$, i.e. it has only finitely many distinct sub-coderivations, then we say that $\mathcal{D}$ is *regular* (or *circular*).

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. Also, from a regular coderivation $\mathcal{D}$ we obtain a *finite* equational program for $f_\mathcal{D}$. In particular, while there are continuum many (non-wellfounded) coderivations, there are only countably many regular ones.

▶ **Example 21.** In Figure 3, $\mathcal{F}(f)$ and $\mathcal{A}(r)$ are the only non-regular coderivations (as long as $\mathcal{G}$, $\mathcal{H}_i$ are regular). Also, $\mathcal{A}(r)$ is R-regular for any $\mathsf{R} \subseteq \mathsf{B}^- - \{0, 1, |\mathsf{cond}|_\Box\}$, since $r(i)$ is computed by just a 0 or 1 step when $r : \mathbb{N} \to \{0,1\}$.

We are now ready to present the non-wellfounded proof systems that will be considered in this paper:

▶ **Definition 22** (CB and nuB)**.** CB is the class of regular progressing safe and left-leaning B^--coderivations. nuB is the class of $\{\mathsf{cond}_\Box, \mathsf{cond}_N, \mathsf{s}_0, \mathsf{s}_1, \mathsf{id}\}$-regular progressing safe and left-leaning B^--coderivations. A two-sorted function $f(\vec{x};\vec{y})$ is CB-*definable* (resp. nuB-*definable*) if there is a coderivation $\mathcal{D} \in \mathsf{CB}$ (resp. $\mathcal{D} \in \mathsf{nuB}$) such that $f_\mathcal{D}(\vec{x};\vec{y}) = f(\vec{x};\vec{y})$.

Recalling Examples 17, 19 and 21, $\mathcal{C}$ is the only coderivation in CB, while $\mathcal{A}(r)$ is an example of coderivation in nuB for any $r : \mathbb{N} \to \{0,1\}$. The system CB was already introduced in [10], where we showed that $\mathsf{CB} = \mathbf{FP}$ (among other results), whereas nuB (read "non-uniform B") is new. The main result of this paper is to show that nuB admits just the right amount of non-wellfoundedness to duly characterise the analogous non-uniform class:

▶ **Theorem 23.** $\mathsf{nuB} = \mathbf{FP}/\mathit{poly}$

The rest of this work is devoted to the proof of this result. In particular, the two directions of the equality are given by Corollary 35 and Corollary 44.

▶ Remark 24 (On proof checking). Let us point out that all conditions on coderivations we have considered so far are *decidable* on regular coderivations. In particular, progressiveness may be decided by reduction to universality of Büchi automata. In the presence of safety, however, it turns out that proof checking becomes easier: checking whether a regular coderivation is in CB is actually decidable in $\mathbf{NL}$ [10, Cor. 32]. Of course, as nuB coderivations are not finitely presented (indeed like $\mathbf{FP}/\mathit{poly}$ programs), such decidability issues are no longer relevant.

4 On relativised characterisations of $\mathbf{FP}/\mathit{poly}$

In this section we consider recursion theoretic characterisations of $\mathbf{FP}/\mathit{poly}$ via relativised function algebras. This will serve not only as a "warm up" to motivate our main characterisation, but will also provide several of the intermediate results necessary to that end. The results of this section are based on textbook techniques and are (presumably) folklore.

4.1 Non-uniformity via resource-bounded oracle machines

A *relation* is a function $r(\vec{x})$ such that we always have $r(\vec{x}) \in \{0,1\}$.

▶ **Definition 25** (Relativised complexity classes)**.** Let R be a set of relations. The class $\mathbf{FP}(R)$ consists of just the functions computable in polynomial time by a Turing machine with access to an oracle for each $r \in R$.

For instance, using this notion of relativised computation, we can define the levels of the functional polynomial hierarchy **FPH** by $\square_1^p := \mathbf{FP}$, $\square_2^p := \mathbf{FP}(\mathbf{NP})$, $\square_3^p := \mathbf{FP}(\Sigma_2^p)$, etc.

Let us write $\mathbb{R} := \{r : \mathbb{N}^k \to \{0,1\} \mid |\vec{x}| = |\vec{y}| \implies r(\vec{x}) = r(\vec{y})\}$. Note that the notation $\mathbb{R}$ is suggestive here, since its elements are essentially maps from lengths/positions to Booleans, and so may be identified with Boolean streams.

▶ **Proposition 26.** $\mathbf{FP}/\mathit{poly} = \mathbf{FP}(\mathbb{R})$.

Proof sketch. For the left-right inclusion, let $p(n)$ be a polynomial and $\mathbf{C} = (C_n)_{n<\omega}$ be a circuit family with each C_n taking n Boolean inputs and having size $< p(n)$. We need to show that the language computed by $\mathbf{C}$ is also computed in $\mathbf{FP}(\mathbb{R})$. Let $c \in \mathbb{R}$ be the function that, on inputs x, y returns the $|y|^{\text{th}}$ bit of $C_{|x|}$. Using this oracle we can compute $C_{|x|}$ by polynomially queries to c, and this may be evaluated as usual using a polynomial-time evaluator in **FP**.

For the right-left inclusion, notice that a polynomial-time machine can only make polynomially many calls to oracles with inputs of only polynomial size. Thus, if $f \in \mathbf{FP}(\mathbb{R})$ then there is some p_f with $f \in \mathbf{FP}(\mathbb{R}^{<p_f})$, where $\mathbb{R}^{<p_f}$ is the restriction of each $r \in \mathbb{R}$ to only its first $p_f(|\vec{x}|)$ many bits. Now, since f can only call a fixed number of oracles from $\mathbb{R}$, we can collect these finitely many polynomial-length prefixes into a single advice string for computation in $\mathbf{FP}/\mathit{poly}$. ◀

4.2 A relativised Bellantoni-Cook characterisation of $\mathrm{FP}/\mathit{poly}$

We shall employ the following writing conventions for the remainder of this work. For a set of (single-sorted) functions F, let us write:

- $F_{1;0}$ for the set of two-sorted functions $f(\vec{x};)$ for each $f(\vec{x}) \in F$;
- $F_{1;1}$ for the set of two-sorted functions $f(\vec{x};\vec{y})$ for each $f(\vec{x},\vec{y}) \in F$.

Given a set F of two-sorted functions, the algebra $\mathsf{B}(F)$ is defined just like B but with additional initial (two-sorted) functions F. Note that, since functions of $\mathsf{B}(F)$ are given by finite programs, they can only depend on finitely many members of F.

▶ **Proposition 27.** $\mathbf{FP}/\mathit{poly} \subseteq \mathsf{B}(\mathbb{R}_{1;0})$

One natural way to prove this result would be to go via $\mathbf{FP}(\mathbb{R})$, in light of Proposition 26. Indeed Bellantoni established foundational results relating $\mathbf{FP}(R)$ and versions of $\mathsf{B}(R)$, for R a set of relations, in [3], but unfortunately the sorting of the corresponding arguments is subtle and does not immediately give the result we are after. For this reason we give a direct proof, that nonetheless inlines some ideas from [3].

Proof of Proposition 27. Let $\mathbf{C} = (C_n)_{n<\omega}$ be a circuit family with each C_n taking n inputs and having size $< p(n)$, for some (monotone) polynomial p. We need to show that the language computed by $\mathbf{C}$ is also computed in $\mathsf{B}(\mathbb{R}_{1;0})$.

First, let $\mathrm{Eval}(x,y)$ evaluate the circuit described by x on the input y. Since $\mathrm{Eval} \in \mathbf{FP}$, we have as standard (e.g. by [4, Lemma 3.2]) a function $\mathrm{Eval}(m;x,y) \in \mathsf{B}$ and a monotone polynomial q such that $|m| \geq q(|x|,|y|) \implies \mathrm{Eval}(m;x,y) = \mathrm{Eval}(x,y)$. Now, in particular, if x is the description of some C_n and $n = |y|$, then also $|x| \leq p(|y|)$, and so $|m| \geq q(p(|y|), |y|) \implies \mathrm{Eval}(m; x, y) = \mathrm{Eval}(x, y)$. Finally, denoting $\overbrace{\mathsf{s}_1 \dots \mathsf{s}_1}^{n} 0$ by 1^n, this means that we have $\mathrm{Eval}(y; x) := \mathrm{Eval}(1^{q(p(|y|),|y|)}; x, y) \in \mathsf{B}$, that in particular evaluates, when x describes $C_{|y|}$, the circuit $C_{|y|}$ on input y.

Now, let $c \in \mathbb{R}_{1;0}$ with $c(y, z;) = |z|^{\text{th}}$ bit of $C_{|y|}$. We show that the function $C(y, z;) = c(y, 0;) \cdot c(y, 1;) \cdot \dots \cdot c(y, 1^{|z|-1};)$ is in $\mathsf{B}(c)$ by the following instance of safe recursion:

$$\begin{aligned} C(y, 0;) &= 0 \\ C(y, \mathsf{s}_i z;) &= \mathsf{cond}(; c(y, z;), \mathsf{s}_0(; C(y, z;)), \mathsf{s}_1(; C(y, z;))) \end{aligned}$$

So we have that $C(y;) := C(y, 1^{p(|y|)};)$ computes the description of $C_{|y|}$. Now we can decide whether y is accepted by $C_{|y|}$ simply by calling the function $\mathrm{Eval}(y; C(y;)) \in \mathsf{B}(c)$. ◀

It turns out that we also have the converse inclusion too. This will be subsumed by our later results but we include it here for the sake of completeness. The key is to establish a general form of Bellantoni and Cook's polymax bounding lemma to account for modulus of continuity as well as growth:

▶ **Lemma 28** (Relational bounding lemma for B). *Let R be a set of two-sorted relations, and suppose $f(R)(\vec{x}; \vec{y}) \in \mathsf{B}(R)$. Then there is a polynomial p_f such that, setting $m_f(\vec{m}, \vec{n}) := p_f(\vec{m}) + \max \vec{n}$, we have:*
- *(Polynomial modulus of growth) $|f(R)(\vec{x}; \vec{y})| < m_f(|\vec{x}|, |\vec{y}|)$*
- *(Polynomial modulus of continuity) $f(R)(\vec{x}; \vec{y}) = f(\lambda |\vec{u}|, |\vec{v}| < m_f(|\vec{x}|, |\vec{y}|).r(\vec{u}; \vec{v}))_{r \in R}(\vec{x}; \vec{y})$*

Using this we may establish:

▶ **Proposition 29** (E.g. see [3]). *Let R be a set of relations.* $\mathsf{B}(R_{1;1}) \subseteq \mathbf{FP}(R)$.

We shall not actually need this result directly in this work, rather recovering (a version of) it from a more refined grand tour of inclusions. However this does lead to the first "implicit" characterisation of $\mathbf{FP}/\mathit{poly}$ of this work:

▶ **Corollary 30.** $\mathsf{B}(\mathbb{R}_{1;0}) = \mathbf{FP}/\mathit{poly}$

5 $\mathbf{FP}/\mathit{poly} \subseteq$ nuB via relativised circular systems

In this section we establish one direction of Theorem 23. In particular, by the end of this section, we will have established the following inclusions,

$$\mathbf{FP}/\mathit{poly} \subseteq \mathsf{B}(\mathbb{R}_{1;0}) \subseteq \mathsf{CB}(\mathbb{R}_{1;0}) \subseteq \mathsf{nuB}$$

where $\mathsf{CB}(F)$ is an extension of CB by new initial sequents for two-sorted functions in F.

5.1 Relativised simulation of B in CB

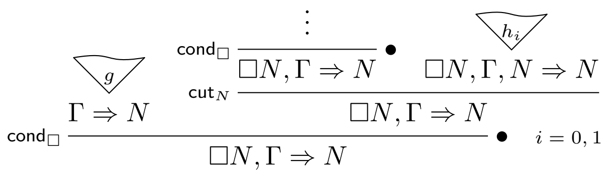
We shall consider "relativised" versions of CB, that may include new initial sequents. Formally:

▶ **Definition 31.** Let F be a set of two-sorted functions. A $\mathsf{B}^-(F)$-coderivation is just a usual B^--coderivation that may use initial sequents of the form $f \, \dfrac{}{\Box N^{n_i}, N^{m_i} \Rightarrow N}$, when $f \in F$ takes n_i normal and m_i safe inputs. We write $\mathsf{CB}(F)$ for the set of CB-coderivations allowing initial sequents from F. The semantics of such coderivations and the notion of $\mathsf{CB}(F)$-definability are as expected, with $f_{\mathcal{D}(F)}$ denoting the induced interpretation of $\mathcal{D}(F) \in \mathsf{CB}(F)$.

Note, again, that since $\mathsf{CB}(F)$ coderivations are regular, they only depend on finitely many members of F. By a modular extension of the result that $\mathsf{B} \subseteq \mathsf{CB}$ from [10], we obtain:

▶ **Proposition 32.** *Let F be a set of two-sorted functions.* $\mathsf{B}(F) \subseteq \mathsf{CB}(F)$.

The proof is simply by structural induction on the definition of a $\mathsf{B}(F)$ function, where the recursion cases are handled by circularity as in [10]. In particular, if f is defined by safe recursion on notation from g, h_0, h_1 then the corresponding CB-coderivation is given by:



g
$\vdots$
h_i
$\mathsf{cond}_\Box$
$\Box N, \Gamma \Rightarrow N$
$\Box N, \Gamma, N \Rightarrow N$
cut_N
$\Gamma \Rightarrow N$
$\Box N, \Gamma \Rightarrow N$
$\mathsf{cond}_\Box$
$\Box N, \Gamma \Rightarrow N$
$\bullet$
$i = 0, 1$

The only new cases in the induction are for an initial function from F, which is simply translated into the appropriate initial sequent.

5.2 Simulating $\mathbb{R}_{1;0}$ oracles in nuB

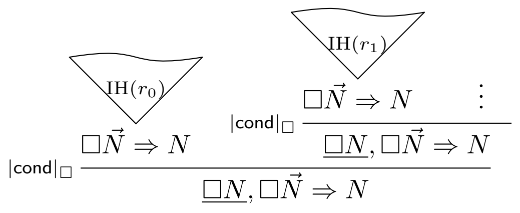
In this subsection we shall establish:

▶ **Proposition 33.** $\mathsf{CB}(\mathbb{R}_{1;0}) \subseteq \mathsf{nuB}$.

By definition of nuB and CB, it suffices to only consider the new initial sequents from $\mathbb{R}_{1;0}$. For this we simply appeal to the following lemma:

▶ **Lemma 34.** *For each $r(\vec{x};) \in \mathbb{R}_{1;0}$, there is a* nuB*-coderivation defining it, in particular using only the rules* $0, 1, |\mathsf{cond}|_\Box$.

Proof. We proceed by induction on the length of $\vec{x}$. When the list is empty, then $r(;)$ is just a Boolean, in which case we can derive it with just the 0 or 1 step. Now, for $r(x, \vec{x};)$ we have:



IH(r_1)
IH(r_0)
$\Box\vec{N} \Rightarrow N$
$\vdots$
$|\mathsf{cond}|_\Box$
$\Box\vec{N} \Rightarrow N$
$\underline{\Box N}, \Box\vec{N} \Rightarrow N$
$|\mathsf{cond}|_\Box$
$\underline{\Box N}, \Box\vec{N} \Rightarrow N$

where $r_i(\vec{x})$ is the function $r(1^i, \vec{x})$ and the coderivations marked $\mathrm{IH}(r_i)$ are obtained by the inductive hypothesis for r_i. ◀

Note that, by putting together Proposition 27, Proposition 32 (setting $F = \mathbb{R}_{1;0}$) and Proposition 33, we now have one half of our main result:

▶ **Corollary 35.** $\mathbf{FP}/\mathit{poly} \subseteq \mathsf{nuB}$

6 nuB as relativised regular coderivations

To facilitate the other direction of Theorem 23, let us first address a form of converse to Proposition 33 above, that duly embeds nuB into a relativised circular system, which we shall rely on in the next section:

▶ **Theorem 36.** $\mathsf{nuB} \subseteq \mathsf{CB}(\mathbb{R}_{1;1})$.

Before giving the proof, we need to first establish some intermediate results:

▶ **Lemma 37.** *If $\mathcal{D}$ is progressing and $\{\mathsf{s}_0, \mathsf{s}_1, \mathsf{id}\}$-free then $f_\mathcal{D}$ is a relation, i.e. $f_\mathcal{D}(\vec{x};\vec{y}) \leq 1$.*

Proof sketch. We proceed by contradiction, always assuming Proposition 16, that progressing coderivations compute total functions.

If $f_\mathcal{D}(\vec{x};\vec{y}) > 1$ then we argue that there is an immediate sub-coderivation $\mathcal{D}'$ of $\mathcal{D}$ and arguments $\vec{x}', \vec{y}'$ such that $f_{\mathcal{D}'}(\vec{x}';\vec{y}') > 1$. Some of the critical cases are:

- If $\mathcal{D} = \mathsf{cut}_N(\mathcal{D}_0, \mathcal{D}_1)$ then $f_\mathcal{D}(\vec{x};\vec{y}) = f_{\mathcal{D}_1}(\vec{x};\vec{y}, f_{\mathcal{D}_0}(\vec{x};\vec{y}))$, and we set $\mathcal{D}' := \mathcal{D}_1$, $\vec{x}' := \vec{x}$ and $\vec{y}' := \vec{y}, f_{\mathcal{D}_0}(\vec{x};\vec{y})$ (since $f_{\mathcal{D}_0}(\vec{x};\vec{y})$ is well-defined). The case for $\mathsf{cut}_\Box$ is similar.
- If $\mathcal{D} = \mathsf{cond}_\Box(\mathcal{D}_0, \mathcal{D}_1, \mathcal{D}_2)$ then $\vec{x} = x, \vec{z}$ with $f_\mathcal{D}(0, \vec{z};\vec{y}) = \mathcal{D}_0(\vec{z};\vec{y})$ and $f_\mathcal{D}(\mathsf{s}_i x', \vec{z};\vec{y}) = f_{\mathcal{D}_{i+1}}(x', \vec{z};\vec{y})$. If $x = 0$ we set $\mathcal{D}' := \mathcal{D}_0$, $\vec{x}' := \vec{z}$, and $\vec{y}' := \vec{y}$. If $x = \mathsf{s}_i x'$ then we set $\mathcal{D}' := \mathcal{D}_{i+1}$, $\vec{x}' := x', \vec{z}$, and $\vec{y}' := \vec{y}$. The cases for cond_N, $|\mathsf{cond}|_N$, and $|\mathsf{cond}|_\Box$ are similar.
- In all other cases $\mathcal{D}$ ends with a unary rule so that $\mathcal{D}'$ is the only immediate sub-coderivation, and $\vec{x}', \vec{y}'$ are determined by the semantics of the rule (cf. Figure 2).

Note that, in the absence of $\mathsf{s}_0, \mathsf{s}_1$, we indeed have that $f'(\vec{x}';\vec{y}') = f(\vec{x};\vec{y}) > 1$ so we may continually apply this process to build up a branch $B = (\mathcal{D} = \mathcal{D}^{(0)}, \mathcal{D}^{(1)}, \mathcal{D}^{(2)}, \dots)$ and arguments $(\vec{x};\vec{y}) = (\vec{x}^{(0)};\vec{y}^{(0)}), (\vec{x}^{(1)};\vec{y}^{(1)}), (\vec{x}^{(2)};\vec{y}^{(2)}), \dots$ such that $f_{\mathcal{D}^{(k)}}(\vec{x}^{(k)};\vec{y}^{(k)}) = f_\mathcal{D}(\vec{x};\vec{y}) > 1$. Observe that B cannot end at an id step, by assumption that $\mathcal{D}$ is id-free. Also, if B ends at a 0 or 1 step we have by construction that $f_\mathcal{D}(\vec{x};\vec{y}) \in \{0, 1\}$, a contradiction. Thus B must be infinite. Since $\mathcal{D}$ is progressing there is a progressing thread along B, say $(\Box N^i)_{i \geq k}$, where each $\Box N^i$ is an occurrence of $\Box N$. Let us examine the values, say x^i, assigned to each $\Box N^i$. Notice that:

- by inspection of the rules and their interpretations from Definition 11, we have that $x^{i+1} \leq x^i$; and,
- if $\Box N^i$ is principal for a $\mathsf{cond}_\Box$ or a $|\mathsf{cond}|_\Box$ step then $x^{i+1} < x^i$.

If follows that $(x^i)_{i \geq k}$ is a non-increasing sequence of natural numbers that does not converge, contradicting the well-ordering property of $\mathbb{N}$. ◀

▶ **Lemma 38.** *If $\mathcal{D}$ is $\{\mathsf{cond}_\Box, \mathsf{cond}_N, \mathsf{id}\}$-free, $|\vec{x}| = |\vec{x}'|$ and $|\vec{y}| = |\vec{y}'|$, then $f_\mathcal{D}(\vec{x};\vec{y}) = f_\mathcal{D}(\vec{x}';\vec{y}')$, whenever $f(\vec{x};\vec{y})$ is well-defined.*

Proof sketch. Being given by an equational program, we have that $f_\mathcal{D}(\vec{x};\vec{y}) = m$ has a (finite) equational derivation for some $m \in \mathbb{N}$, by assumption that it is well-defined (cf. Remark 12). Replacing $\vec{x}, \vec{y}$ by $\vec{x}', \vec{y}'$ in this derivation yields $f_\mathcal{D}(\vec{x}';\vec{y}') = m$ too. The only critical cases are the steps $|\mathsf{cond}|_N$, $|\mathsf{cond}|_\Box$, whose semantics only depend on the length of their arguments. ◀

Putting the two above Lemmata together we have:

▶ **Proposition 39.** *If $\mathcal{D}$ is progressing and $\{\mathsf{cond}_\Box, \mathsf{cond}_N, \mathsf{s}_0, \mathsf{s}_1, \mathsf{id}\}$-free, then $f_\mathcal{D} \in \mathbb{R}_{1;1}$.*

Now we can prove Theorem 36:

Proof sketch. Let $\mathcal{D}$ be a nuB-coderivation and let V be the set of minimal nodes ν such that $\mathcal{D}_\nu$ is $\{\mathsf{cond}_\Box, \mathsf{cond}_N, \mathsf{s}_0, \mathsf{s}_1, \mathsf{id}\}$-free, and so by Proposition 39 we have that each $f_{\mathcal{D}_\nu} \in \mathbb{R}_{1;1}$.

Now, let $\mathcal{D}^V$ be obtained from $\mathcal{D}$ by simply deleting each sub-coderivation $\mathcal{D}_\nu$, for $\nu \in V$, and construing each of their conclusions as new initial sequents. By definition of nuB, note that $\mathcal{D}^V$ is now a coderivation in $\mathsf{CB}(f_{\mathcal{D}_\nu})_{\nu \in V} \subseteq \mathsf{CB}(\mathbb{R}_{1;1})$ and we are done. ◀

7 nuB $\subseteq$ FP/*poly*: a relativised algebra subsuming circular typing

The final part of our chain of inclusions requires us to translate (relativised) circular coderivations into an appropriate function algebra. The idea is that, in the presence of safety, one can reduce circularity to a form of recursion on "permutations of prefixes" that nonetheless remains feasible. This was (one of) the main result(s) of [10] and, fortunately, we are able to import those results accounting only for additional initial relations.

7.1 Safe recursion on permutations of prefixes

Let us write $\vec{x} \subseteq \vec{y}$ if $\vec{x}$ is a permutation of prefixes of $\vec{y}$, i.e. $\vec{x} = x_0, \dots, x_{n-1}$ and $\vec{y} = y_0, \dots, y_{n-1}$ and there is a permutation $\pi : [n] \to [n]$ s.t. each x_i is a prefix of $y_{\pi i}$. We shall write $\vec{x} \subset \vec{y}$ if for at least one $i < n$ we have that x_i is a strict prefix of $y_{\pi i}$. Note in particular that $\subset$ is a well-founded pre-order so admits an induction and recursion principles.

To formulate recursion over well-founded relations it is convenient to employ (two-sorted) oracles as placeholders for recursive calls. Due to the necessary constraints on composition, we shall formally distinguish these oracles (metavariables, a, b, etc.) from additional initial functions (metavariables f, g etc. until now).

▶ **Definition 40.** Let F be a set of two-sorted functions. The algebra $\mathsf{B}^{\subset}(F, \vec{a})$ is the smallest class of two-sorted functions containing,

- all the initial functions of B;
- each (two-sorted) function a_i among $\vec{a}$;
- each (two-sorted) function $f \in F$;

and closed under:

- (Relativised safe composition)
  - if $g(\vec{x};\vec{y}) \in \mathsf{B}^{\subset}(F, \vec{a})$ and $h(\vec{x};\vec{y}, y) \in \mathsf{B}^{\subset}(F, \vec{a})$ then $f(\vec{x};\vec{y}) := h(\vec{x};\vec{y}, g(\vec{x};\vec{y})) \in \mathsf{B}^{\subset}(F, \vec{a})$;
  - if $g(\vec{x};) \in \mathsf{B}^{\subset}(F)$ and $h(\vec{x}, x;\vec{y}) \in \mathsf{B}^{\subset}(F, \vec{a})$ then $f(\vec{x};\vec{y}) := h(\vec{x}, g(\vec{x};);\vec{y}) \in \mathsf{B}^{\subset}(F, \vec{a})$;
- (Safe recursion on $\subset$)
  if $h(a)(\vec{x};\vec{y}) \in \mathsf{B}^{\subset}(F, a, \vec{a})$ then $f(\vec{x};\vec{y}) := h(\lambda\vec{u} \subset \vec{x}, \lambda\vec{v} \subseteq \vec{y}. f(\vec{u};\vec{v}))(\vec{x};\vec{y}) \in \mathsf{B}^{\subset}(F, \vec{a})$.

To be clear, the "guarded" abstraction notation above is formally defined as

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

$\mathsf{B}^{\subset}(\varnothing, \vec{a})$ is the same as the notion $\mathsf{B}^{\subset}(\vec{a})$ from [10]. Note in particular the distinction between F and $\vec{a}$ in the safe composition scheme: when composing along a normal parameter (second line), the function $g(\vec{x};)$ must not contain any oracles among $\vec{a}$.

Adapting the *Bounding Lemma* from [10, Lemma 38] to account for further initial relations gives:

▶ **Lemma 41** (Relational Bounding lemma). *Let R be a set of two-sorted relations and $f(R)(\vec{x};\vec{y}) \in \mathsf{B}^{\subset}(R)$. There is a polynomial $p_f(\vec{n})$ such that, writing $m_f(\vec{x},\vec{y}) = p_f(|\vec{x}|) + \max|\vec{y}|$, we have:*

- $|f(R)(\vec{x};\vec{y})| < m_f(\vec{x},\vec{y})$
- $f(R)(\vec{x};\vec{y}) = f(\lambda|\vec{u}_r|, |\vec{v}_r| < m_f(\vec{x},\vec{y}).r(\vec{u}_r;\vec{v}_r))_{r\in R}(\vec{x};\vec{y})$

The first point is common to implicit complexity, being essentially Bellantoni and Cook's "polymax bounding lemma" from [4]. The second point expresses a dual property: while the first bounds the modulus of *growth*, the second bounds the modulus of *continuity*.

Here it is important the the new initial functions are relations, or at least that they have constant/limited growth rate. In fact, for the proof, cf. [10], one needs a more complicated statement accounting for growth properties of the intermediate oracles $\vec{a}$ used for recursion, even though we only ultimately need the statement above for our purposes, once all such oracles $\vec{a}$ are "discharged".

Using the Bounding Lemma we have from [10] (again accounting for further initial relations) the main characterisation result for $\mathsf{B}^{\subset}$:

▶ **Proposition 42** (Relativised characterisation). *For a set R of relations,* $\mathsf{B}^{\subset}(R_{1;1}) \subseteq \mathbf{FP}(R)$.

The main point for proving this result is that the graph of $\subseteq$ is relatively small, in particular for each $\vec{y}$ there are only polynomially many $\vec{x} \subseteq \vec{y}$.[1] So we can calculate a function $f(\vec{x};\vec{y}) \in \mathsf{B}^{\subset}(R)$ simply by polynomial-time induction (at the meta level) on $\subset$, storing all previous values in a lookup table. This table will have only polynomially many entries, by the previous observation about the size of the graph of $\subseteq$, and each entry will have only polynomial size by the Bounding Lemma.

7.2 $\mathsf{CB}(\mathbb{R}_{1;1}) \subseteq \mathsf{B}^{\subset}(\mathbb{R}_{1;1})$: from circular proofs to recursive functions

The point of $\mathsf{B}^{\subset}$ in [10] was to play the role of a target algebra to translate circular coderivations into. The *Translation Lemma* from that work [10, Lemma 47], accounting for further initial relations, gives:

▶ **Lemma 43** (Relativised translation). *Let R be a set of relations.* $\mathsf{CB}(R_{1;1}) \subseteq \mathsf{B}^{\subset}(R_{1;1})$.

Note that we specialise the statement above only to sets of relations to avoid size issues potentially caused be new initial funtions of arbitrary growth rate. In fact, this proof requires closure of $\mathsf{B}^{\subset}(F, \vec{a})$ under a *simultaneous* version of its recursion scheme, upon which a careful translation from circular coderivations in "cycle normal form" (see, e.g., [7, Definition 6.2.1]) to an equational specification can be duly resolved in $\mathsf{B}^{\subset}$.

Now by setting $R = \mathbb{R}$, we have the following consequence of Proposition 42:

▶ **Corollary 44.** $\mathsf{nuB} \subseteq \mathbf{FP}/\mathit{poly}$

Proof. We have $\mathsf{nuB} \subseteq \mathsf{CB}(\mathbb{R}_{1;1})$ by Theorem 36, $\mathsf{CB}(\mathbb{R}_{1;1}) \subseteq \mathsf{B}^{\subset}(\mathbb{R}_{1;1})$ by Lemma 43, $\mathsf{B}^{\subset}(\mathbb{R}_{1;1}) \subseteq \mathbf{FP}(\mathbb{R})$ by Proposition 42, and finally $\mathbf{FP}(\mathbb{R}) \subseteq \mathbf{FP}/\mathit{poly}$ by Proposition 26. ◀

Along with Corollary 35, we have now established both directions of our main result Theorem 23 that $\mathsf{nuB} = \mathbf{FP}/\mathit{poly}$, hence completing the proof.

8 Conclusions

In this work we presented the two-sorted non-wellfounded proof system nuB and proved that it characterises the complexity class $\mathbf{FP}/\mathit{poly}$. Our results build on previous work [10], where we defined the cyclic proof systems CB and CNB capturing, respectively, $\mathbf{FP}$ and

[1] Of course, this polynomial depends on the length of $\vec{y}$, but for a given function of the algebra this is some global constant.

FELEMENTARY [10]. The system nuB is obtained from CB by associating non-uniformity in computation to a form of non-wellfoundedness in proof theory. To establish the characterisation theorems, we also formalised some (presumably) folklore results on relativised function algebras for **FP**/*poly*.

For future research, the first author is investigating non-wellfounded approaches to **FP**/*poly* in the setting of linear logic [19]. In particular, we are studying a non-wellfounded version of Mazza's Parsimonious Logic [28], a variant of linear logic where the exponential modality ! satisfies Milner's law ($!A \simeq !A \otimes A$). This provides a natural computational interpretation of formulas $!A$ as types of streams on A. Mazza showed in [29] that Parsimonious Logic can be used to capture **P**/*poly* using *wellfounded* proofs that are essentially *infinitely branching*. We conjecture that a similar characterisation can be obtained in a non-wellfounded (and finitely branching) setting, using ideas from this work.

Another direction is to explore applications of the results of this paper to probabilistic complexity. In particular, we aim to study fragments of nuB modelling the class **BPP** (bounded-error probabilistic polynomial time), essentially by leveraging on well-known derandomisation methods showing the inclusion of **BPP** in **FP**/*poly*, and hence in **FP**($\mathbb{R}$) (see Proposition 26). A challenging aspect of this task is to obtain characterisation results that are entirely in the style of ICC, since **BPP** is defined by explicit (error) bounds, as observed in [25]. We suspect that nuB represents the right framework for investigating fully implicit characterisations of this class, where additional proof-theoretic conditions can be introduced to restrict computationally the access to oracles and, consequently, to model bounded-error probabilistic computation.

As [10] also established a system CNB for **FELEMENTARY**, it would be pertinent to ask whether ideas in this work can be applied to CNB to characterise **FELEMENTARY**/*poly*, i.e., the class of functions computable in elementary time by a Turing machine with access to a polynomial advice. Unfortunately, the modulus of continuity established for CNB in [10] is super-polynomial (indeed elementary), meaning that the same technique, a priori, would not restrict computation to only polynomial advice. Consideration of this issue is left to future research.

References

1 Sanjeev Arora and Boaz Barak. *Computational Complexity – A Modern Approach.* Cambridge University Press, 2009. URL: `http://www.cambridge.org/catalogue/catalogue.asp?isbn=9780521424264`.

2 David Baelde, Amina Doumane, and Alexis Saurin. Infinitary proof theory: the multiplicative additive case. In Jean-Marc Talbot and Laurent Regnier, editors, *25th EACSL Annual Conference on Computer Science Logic, CSL 2016, August 29 – September 1, 2016, Marseille, France*, volume 62 of *LIPIcs*, pages 42:1–42:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. `doi:10.4230/LIPIcs.CSL.2016.42`.

3 Stephen Bellantoni. Predicative recursion and the polytime hierarchy. In Peter Clote and Jeffrey B. Remmel, editors, *Feasible Mathematics II*, pages 15–29, Boston, MA, 1995. Birkhäuser Boston.

4 Stephen Bellantoni and Stephen Cook. A new recursion-theoretic characterization of the polytime functions (extended abstract). In *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing*, STOC '92, pages 283–293, New York, NY, USA, 1992. Association for Computing Machinery. `doi:10.1145/129712.129740`.

5 Stefano Berardi and Makoto Tatsuta. 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*, pages 1–12. IEEE Computer Society, 2017. `doi:10.1109/LICS.2017.8005114`.

6 Stefano Berardi and Makoto Tatsuta. Classical system of Martin-Lof's inductive definitions is not equivalent to cyclic proofs. *Log. Methods Comput. Sci.*, 15(3), 2019. doi:10.23638/LMCS-15(3:10)2019.

7 James Brotherston. *Sequent calculus proof systems for inductive definitions.* PhD thesis, University of Edinburgh, 2006. PhD thesis.

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

9 Samuel R. Buss. Chapter i – an introduction to proof theory. In Samuel R. Buss, editor, *Handbook of Proof Theory*, volume 137 of *Studies in Logic and the Foundations of Mathematics*, pages 1–78. Elsevier, 1998. doi:10.1016/S0049-237X(98)80016-5.

10 Gianluca Curzi and Anupam Das. Cyclic implicit complexity. *CoRR*, abs/2110.01114, 2021. To appear in proceedings of *LICS 2022*. arXiv:2110.01114.

11 Anupam Das. A circular version of Gödel's T and its abstraction complexity. *CoRR*, abs/2012.14421, 2020. arXiv:2012.14421.

12 Anupam Das. On the logical complexity of cyclic arithmetic. *Log. Methods Comput. Sci.*, 16(1), 2020. doi:10.23638/LMCS-16(1:1)2020.

13 Anupam Das. On the logical strength of confluence and normalisation for cyclic proofs. In Naoki Kobayashi, editor, *6th International Conference on Formal Structures for Computation and Deduction, FSCD 2021, July 17-24, 2021, Buenos Aires, Argentina (Virtual Conference)*, volume 195 of *LIPIcs*, pages 29:1–29:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.FSCD.2021.29.

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

15 Anupam Das and Damien Pous. Non-Wellfounded Proof Theory For (Kleene+Action) (Algebras+Lattices). In Dan Ghica and Achim Jung, editors, *27th EACSL Annual Conference on Computer Science Logic (CSL 2018)*, volume 119 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 19:1–19:18, Dagstuhl, Germany, 2018. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CSL.2018.19.

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

17 Abhishek De and Alexis Saurin. Infinets: The parallel syntax for non-wellfounded proof-theory. In Serenella Cerrito and Andrei Popescu, editors, *Automated Reasoning with Analytic Tableaux and Related Methods – 28th International Conference, TABLEAUX 2019, London, UK, September 3-5, 2019, Proceedings*, volume 11714 of *Lecture Notes in Computer Science*, pages 297–316. Springer, 2019. doi:10.1007/978-3-030-29026-9_17.

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

19 Jean-Yves Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987. doi:10.1016/0304-3975(87)90045-4.

20 Emmanuel Hainry, Damiano Mazza, and Romain Péchoux. Polynomial time over the reals with parsimony. In Keisuke Nakano and Konstantinos Sagonas, editors, *Functional and Logic Programming – 15th International Symposium, FLOPS 2020, Akita, Japan, September 14-16, 2020, Proceedings*, volume 12073 of *Lecture Notes in Computer Science*, pages 50–65. Springer, 2020. doi:10.1007/978-3-030-59025-3_4.

21 Martin Hofmann. A mixed modal/linear lambda calculus with applications to Bellantoni-Cook safe recursion. In Mogens Nielsen and Wolfgang Thomas, editors, *Computer Science Logic, 11th International Workshop, CSL '97, Annual Conference of the EACSL, Aarhus, Denmark, August 23-29, 1997, Selected Papers*, volume 1414 of *Lecture Notes in Computer Science*, pages 275–294. Springer, 1997. doi:10.1007/BFb0028020.

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

23 Ker-I Ko. *Complexity Theory of Real Functions*. Birkhauser Boston Inc., USA, 1991.

24 Denis Kuperberg, Laureline Pinault, and Damien Pous. Cyclic proofs, system T, and the power of contraction. *Proc. ACM Program. Lang.*, 5(POPL):1–28, 2021. doi:10.1145/3434282.

25 Ugo Dal Lago and Paolo Parisen Toldin. A higher-order characterization of probabilistic polynomial time. *Inf. Comput.*, 241:114–141, 2015. doi:10.1016/j.ic.2014.10.009.

26 Daniel Leivant. A foundational delineation of computational feasiblity. In *Proceedings of the Sixth Annual Symposium on Logic in Computer Science (LICS '91), Amsterdam, The Netherlands, July 15-18, 1991*, pages 2–11. IEEE Computer Society, 1991. doi:10.1109/LICS.1991.151625.

27 Damiano Mazza. Non-uniform polytime computation in the infinitary affine lambda-calculus. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming – 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II*, volume 8573 of *Lecture Notes in Computer Science*, pages 305–317. Springer, 2014. doi:10.1007/978-3-662-43951-7_26.

28 Damiano Mazza. Simple parsimonious types and logarithmic space. In Stephan Kreutzer, editor, *24th EACSL Annual Conference on Computer Science Logic, CSL 2015, September 7-10, 2015, Berlin, Germany*, volume 41 of *LIPIcs*, pages 24–40. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.CSL.2015.24.

29 Damiano Mazza and Kazushige Terui. Parsimonious types and non-uniform computation. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming – 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, volume 9135 of *Lecture Notes in Computer Science*, pages 350–361. Springer, 2015. doi:10.1007/978-3-662-47666-6_28.

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

31 Damian Niwiński and Igor Walukiewicz. Games for the μ-calculus. *Theoretical Computer Science*, 163(1-2):99–116, 1996.

32 Alex Simpson. Cyclic arithmetic is equivalent to peano arithmetic. In Javier Esparza and Andrzej S. Murawski, editors, *Foundations of Software Science and Computation Structures – 20th International Conference, FOSSACS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, volume 10203 of *Lecture Notes in Computer Science*, pages 283–300, 2017. doi:10.1007/978-3-662-54458-7_17.