Non-wellfounded parsimonious proofs
and non-uniform complexity

Matteo Acclavio
University of Southern Denmark
macclavio@gmail.com

Gianluca Curzi
University of Gothenburg
gianluca.curzi@gu.se

Giulio Guerrieri
Aix Marseille Université
giulio.guerrieri@lis-lab.fr

ABSTRACT

In this paper we investigate the complexity-theoretical aspects of cyclic and non-wellfounded proofs in the context of parsimonious logic, a variant of linear logic where the exponential modality ! is interpreted as a constructor for streams over finite data. We present non-wellfounded parsimonious proof systems capturing the classes **FP** and **FP**/poly. Soundness is established via a polynomial modulus of continuity for continuous cut-elimination. Completeness relies on an encoding of polynomial Turing machines with advice.

As a byproduct of our proof methods, we establish a series of characterisation results for various finitary proof systems.

1 INTRODUCTION

In its modern guise, *non-wellfounded proof theory* emerged for the first time in the context of the modal μ-calculus [23, 43]. Since then, this area of proof-theory has provided a promising theoretical framework for studying least and greatest fixed points, hence for reasoning about induction and coinduction. What is more, its applications have spanned, over the years, a number of rather diverse topics, such as predicate logic [9, 10], algebras [21, 22], arithmetic [8, 19, 45], proofs-as-programs interpretations [4, 18, 20, 24, 33], and continuous cut-elimination [26, 41].

Non-wellfounded proof-theory studies proofs whose underlying tree structure is possibly infinite (but finitely branching). In this setting logical consistency is guaranteed by appropriate global proof-theoretic conditions, called *progressing criteria*. In particular, those non-wellfounded proofs with a *regular* tree structure have received special attention in the literature because they admit a finite description, typically based on cyclic directed graphs. Because of their graph-theoretic representation, these regular proofs are commonly named *circular* or *cyclic*.

In [18, 20, 33] non-wellfounded proof-theory has been investigated from the perspective of the *Curry-Howard correspondence*

paradigm, where proofs are interpreted as (functional) programs, and program execution is given in terms of cut-elimination. Non-wellfounded proofs can be understood as programs defined by a possibly infinite list of instructions, where the progressing criterion ensures *totality*, i.e., that those programs are always well-defined on all arguments. On the other hand, the regularity condition on proof trees has a natural counterpart in the notion of *uniformity*: regular proofs can be properly regarded as programs, i.e. as finite sets of machine instructions, thus having a "computable" behaviour.

In [12] this computational reading of non-wellfounded proofs is extended to the realm of *computational complexity*, introducing circular proof systems capturing 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), i.e., the study of machine-free and bound-free characterisations of complexity classes. More specifically, these circular proof systems are based on Bellantoni and Cook's algebra of functions for safe recursion [7], one of the cornerstones of ICC. These results have been generalized in [14] to capture the class of functions computable in polynomial time by Turing machines with access to *polynomial advice* (**FP**/poly) or, equivalently, computable by non-uniform families of polynomial-size circuits [3]. Specifically, *non-uniform complexity* is modeled by more permissive non-wellfounded proof systems (compared to circular proof systems), obtained by weakening the regularity condition, hence relaxing finite presentability of proofs[1].

In this paper we take an alternative route to non-wellfounded approaches for ICC, which is based on *linear logic* [30]. Linear logic (LL) is a refinement of both classical and intuitionistic logic that allows a better control over computational resources thanks to the so-called *exponential modalities* (denoted by ! and ?), which mark the distinction between those assumptions that can be used linearly (that is, exactly once), and the ones that are reusable at will. According to the Curry-Howard reading of linear logic, these modalities introduce non-linearity in functional programs: a proof of the linear implication $!A \multimap B$ is interpreted as a program returning an output of type B using an arbitrary number of times an input of type A.

Linear logic has inspired a variety of methods for taming complexity. The central idea is to weaken the exponential rules for inducing a bound on cut-elimination, which reduces the computational strength of the system. These restricted systems of linear logic are called "light logics". Examples are *soft linear logic* [34] or *light linear logic* [29] for **FP**, and *elementary linear logic* [5, 16] for

---

[1]Note that **FP**/poly includes *undecidable problems*, and so cannot be characterised by purely circular proof systems, which typically represent only computable functions.

FELEMENTARY. Light logics are typically endowed with second-order quantifiers, which allow for a direct encoding of (resource-bounded) Turing machines, the crucial step for proving completeness w.r.t. a complexity class.

Continuing this tradition, in a series of papers [38–40] Mazza introduced *parsimonious logic* (PL), a variant of linear logic (defined in a type-theoretical fashion) where the exponential modality ! satisfies Milner's law (i.e., $!A \multimap A \otimes !A$) and invalidates the implications $!A \multimap !!A$ (*digging*) and $!A \multimap !A \otimes !A$ (*contraction*). In parsimonious logic, a proof of $!A$ can be interpreted as a *stream* over proofs of A, i.e., as a greatest fixed point. The linear implications $A \otimes !A \multimap !A$ (*co-absorption*) and $!A \multimap A \otimes !A$ (*absorption*), which form the two directions of Milner's law, can be computationally read as the *push* and *pop* operations on streams. In particular, in [40] Mazza and Terui presented *non-uniform parsimonious logic* (nuPL), an extension of PL equipped with an infinitely branching rule ib!p (see Figure 2) that constructs a stream $(\mathcal{D}_{f(0)}, \mathcal{D}_{f(1)}, \ldots, \mathcal{D}_{f(n)}, \ldots)$ of type $!A$ from a finite set of proofs $\mathcal{D}_1, \ldots, \mathcal{D}_n$ of A and a (possibly non-recursive) function $f : \mathbb{N} \to \{1, \ldots, n\}$.

The fundamental result of [40] is that, when endowed with restricted second-order quantifiers, the logic PL (resp. nuPL) characterises the class **P** of problems decidable in polynomial time (resp. the class **P**/poly of problems decidable by polynomial size families of circuits)[2]. On the one hand, the infinitely branching rule can be used to encode streams, hence to model Turing machines querying an *advice* [3]; on the other hand, the absence of digging and contraction induces a polynomial bound on normalisation.

The analysis of parsimonious logic conducted in [38–40] reveals that fixed point definitions of the exponentials are better behaving when digging and contraction are discarded. However, these results rely on the co-absorption rule (?b in Figure 2) which is not admissible in LL. Proper subsystems of LL (free of the co-absorption rule) admitting a stream-based interpretation of the exponentials have been provided in [2], where the authors define *parsimonious linear logic* (PLL) and its non-uniform version, called *non-uniform parsimonious linear logic* (nuPLL). Furthermore, the authors recast PLL and nuPLL in a non-wellfounded framework by identifying appropriate global conditions that duly reflect the proof-theoretic features of these systems. As a result, they introduce *regular parsimonious linear logic* (rPLL), defined in terms of regular non-wellfounded proofs, and *weakly regular parsimonious linear logic* (wrPLL), where regularity is relaxed to model non-uniform computation. The main contribution of [2] is a *continuous cut-elimination theorem* for rPLL and wrPLL, i.e., a cut-elimination result in a non-wellfounded setting.

Contributions. We consider second-order extensions of the proof systems presented in [2], establishing the characterizations below:

- wrPLL$_2^\infty$ (and nuPLL$_2$) characterise **FP**/poly;
- rPLL$_2^\infty$ (and PLL$_2$) characterise **FP**.

The interconnections between our results are summarised in Figure 1. The key contribution is the polynomial modulus of continuity on cut-elimination for wrPLL$_2^\infty$ and rPLL$_2^\infty$, from which we infer that wrPLL$_2^\infty$ is sound for **FP**/poly, and that rPLL$_2^\infty$ is sound for **FP**. Completeness requires a series of intermediate steps. We first introduce a type system (nuPTA$_2$) implementing a form of stream-based
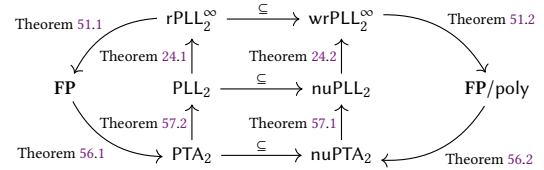

Figure 1: Grand tour diagram of the main results.

computation. Then we describe an encoding of polynomial time Turing machines with (polynomial) advice within nuPTA$_2$, by adapting standard methods from [28, 37] to the setting of non-uniform computation. This allows us to prove that nuPTA$_2$ is complete for **FP**/poly. Thirdly, we define a translation from nuPTA$_2$ to nuPLL$_2$. Finally, we show that computation over strings in nuPLL$_2$ can be simulated within wrPLL$_2^\infty$. A similar completeness argument can be restated for rPLL$_2^\infty$ and PLL$_2$.

On a technical side, the present paper contributes to the previous literature on parsimonious logic [38–40] in many directions. First, since our systems are free of the co-absorption rule, Theorem 58 establishes completeness without a "push" operation on streams, unlike [40]. Secondly, we generalise the characterisation of classes of problems given in [40] (i.e., **P**/poly and **P**) to classes of functions (i.e., **FP**/poly and **FP**), and put them into the realm of (non-wellfounded) LL. Last, the advantage of wrPLL$_2^\infty$ over the system nuPL is that non-wellfounded proofs replace the infinitely branching rule ib!p, avoiding the introduction of the constant growth-rate function $f : \mathbb{N} \to \{1, \ldots, n\}$ and making our characterisations more "implicit", thus closer to ICC.

Outline of the paper. Section 2 recalls some preliminaries on linear logic, non-wellfounded proofs and non-uniform complexity. In Section 3 we introduce parsimonious linear logic and the inductive proof systems PLL$_2$, nuPLL$_2$. In Section 4 we define their non-wellfounded counterpart, i.e., rPLL$_2^\infty$ and wrPLL$_2^\infty$. Finally, soundness and completeness results are discussed in Sections 5 and 6.

Due to space constraints, most of our proofs and technical development is devolved to appendices.

2 PRELIMINARY NOTIONS

In this section we recall some basic notions from (non-wellfounded) proof theory, fixing the notation that will be adopted in this paper.

2.1 Derivations and coderivations

We assume that the reader is familiar with the syntax of sequent calculus, e.g. [46]. Here we specify some conventions adopted to simplify the content of this paper.

We consider (**sequent**) **rules** of the form $r \dfrac{}{\Gamma}$ or $r \dfrac{\Gamma_1}{\Gamma}$ or $r \dfrac{\Gamma_1 \quad \Gamma_2}{\Gamma}$, and we refer to the sequents Γ_1 and Γ_2 as the **premises**, and to the sequent Γ as the **conclusion** of the rule r. To avoid technicalities of the sequents-as-lists presentation, we follow [4] and we consider **sequents** as *sets of occurrences of formulas* from a given set of formulas. In particular, when we refer to a formula in a sequent we always consider a *specific occurrence* of it.

Definition 1. A (binary, possibly infinite) **tree** $\mathcal{T}$ is a subset of words in $\{1, 2\}^*$ that contains the empty word ϵ (the **root** of $\mathcal{T}$) and is *ordered-prefix-closed* (i.e., if $n \in \{1, 2\}$ and $vn \in \mathcal{T}$, then $v \in \mathcal{T}$, and if moreover $v2 \in \mathcal{T}$, then $v1 \in \mathcal{T}$). The elements of $\mathcal{T}$ are called **nodes** and their **height** is the length of the word. A **child** of $v \in \mathcal{T}$ is any $vn \in \mathcal{T}$ with $n \in \{1, 2\}$. The **prefix order** is a partial order $\leq_{\mathcal{T}}$ on $\mathcal{T}$ defined by: for any $v, v' \in \mathcal{T}$, $v \leq_{\mathcal{T}} v'$ if $v' = vw$ for some $w \in \{1, 2\}^*$. A maximal element of $\leq_{\mathcal{T}}$ is a **leaf** of $\mathcal{T}$. A **branch** of $\mathcal{T}$ is a set $\mathcal{B} \subseteq \mathcal{T}$ such that $\epsilon \in \mathcal{B}$ and if $w \in \mathcal{B}$ is not a leaf of $\mathcal{T}$ then w has exactly one child in $\mathcal{B}$.

A **coderivation** over a set of rules $\mathcal{S}$ is a labeling $\mathcal{D}$ of a tree $\mathcal{T}$ by sequents such that if v is a node of $\mathcal{T}$ with children $v_1, \dots, v_n$ (with $n \in \{0, 1, 2\}$), then there is an occurrence of a rule r in $\mathcal{S}$ with conclusion the sequent $\mathcal{D}(v)$ and premises the sequents $\mathcal{D}(v_1), \dots, \mathcal{D}(v_n)$. The **height** of r in $\mathcal{D}$ is the height of $v \in \mathcal{T}$ such that $\mathcal{D}(v)$ is the conclusion of r. The **conclusion** of $\mathcal{D}$ is the sequent $\mathcal{D}(\epsilon)$. If v is a node of the tree, the **sub-coderivation** of $\mathcal{D}$ rooted at v is the coderivation $\mathcal{D}_v$ defined by $\mathcal{D}_v(w) = \mathcal{D}(vw)$.

It is **regular** if it has finitely many distinct sub-coderivations; it is **non-wellfounded** if it labels an infinite tree, and it is a **derivation** (with **size** $|\mathcal{D}| \in \mathbb{N}$) if it labels a finite tree (with $|\mathcal{D}|$ nodes).

Regular coderivations (often called circular or cyclic) can be represented as *finite* directed (possibly cyclic) graphs: a cycle is created by linking the roots of two identical subcoderivations.

Definition 2. Let $\mathcal{D}$ be a coderivation labeling a tree $\mathcal{T}$. A **bar** (resp. **prebar**) of $\mathcal{D}$ is a set $\mathcal{V} \subseteq \mathcal{T}$ where:

- any (resp. infinite) branch of $\mathcal{T}$ contains a node in $\mathcal{V}$;
- nodes in $\mathcal{V}$ are pairwise $\leq_{\mathcal{T}}$-incomparable.

2.2 Non-uniform complexity classes

Our goal is a proof theoretic characterisation of the complexity class **FP**/poly [3], i.e., the class of functions computable in polynomial time (with respect to the length of the input) by a Turing machine having access to a "polynomial amount of advice" (determined only by the length of the input). If **FP** is class of functions computable in polynomial time by a Turing machine, **FP**/poly is defined as follows.

Definition 3. **FP**/poly is the class of functions $f(\vec{x})$ for which, for all $n \in \mathbb{N}$, there is a string (called the **advice**) α_n of length polynomial in n and $f'(y, \vec{x}) \in$ **FP** such that $f(\vec{x}) = f'(\alpha_{|\vec{x}|}, \vec{x})$.

FP/poly extends **FP** and contains some incomputable functions, for instance the characteristic function of undecidable unary languages [3, Example 6.4]. The class **FP**/poly can be also defined in terms of non-uniform families of circuits.

THEOREM 4 ([3], THM. 6.11). *A function f is in* **FP**/poly *iff there is polynomial-size familiy of circuits computing f.*

We adopt a different presentation of **FP**/poly that eases the proof of completeness. A *relation* is a function $r \colon \mathbb{N}^k \to \{0, 1\}$ with $k \in \mathbb{N}$.

Definition 5. Let R be a set of relations. The class **FP**(R) consists of just the functions computable in polynomial time by a Turing machine with access to an oracle for each relation $r \in R$.

Let $\mathbb{R} := \{r \colon \mathbb{N}^k \to \{0, 1\} \mid \exists k \in \mathbb{N}, |\vec{x}| = |\vec{y}| \implies r(\vec{x}) = r(\vec{y})\}$. Note that the notation $\mathbb{R}$ is suggestive here, since its elements

are maps from lengths/positions to Booleans, and so they may be identified with Boolean streams.

Proposition 6 (See, e.g., [14])**.** **FP**/poly = **FP**$(\mathbb{R})$.

3 2ND ORDER PARSIMONIOUS LINEAR LOGIC

In this paper we consider the set of *formulas* for second-order multiplicative-exponential linear logic with units (MELL$_2$). These are generated by a countable set of propositional variables $\mathcal{A} = \{X, Y, \dots\}$ using the following grammar:

$$A ::= X \mid X^{\perp} \mid A \otimes A \mid A \,\mathcal{B}\, A \mid {!}A \mid {?}A \mid 1 \mid \perp \mid \forall X.A \mid \exists X.A$$

A $!$-*formula* (resp. $?$-*formula*) is a formula of the form $!A$ (resp. $?A$). We denote by FV(A) the set of propositional variables occurring free in A, and by $A[B/X]$ the standard meta-level capture-avoiding substitution of B for the free occurrences of the propositional variable X in A. *Linear negation* $(\cdot)^{\perp}$ is defined by De Morgan's laws $(A^{\perp})^{\perp} = A$, $(A \otimes B)^{\perp} = A^{\perp} \,\mathcal{B}\, B^{\perp}$, $(!A)^{\perp} = ?A^{\perp}$, $(1)^{\perp} = \perp$, and $(\forall X.A)^{\perp} = \exists X.A^{\perp}$, while *linear implication* is $A \multimap B := A^{\perp} \,\mathcal{B}\, B$.

Definition 7. **Second-order parsimonious linear logic**, noted PLL$_2$, is the set of rules in Figure 2 on the left, i.e., axiom (ax), cut (cut), *tensor* ($\otimes$), *par* ($\mathcal{B}$), *one* (1), *bottom* ($\perp$), *functorial promotion* (f!p), *weakening* (?w), *absorption* (?b), *(second-order) universal quantifier* ($\forall$), *(second-order) existential quantifier* ($\exists$). Rules ax, $\otimes$, $\mathcal{B}$, 1 and $\perp$ are *multiplicative*, rules f!p, ?w and ?b are *exponential*, rules $\forall$ and $\exists$ are *second-order*[3]. The set of derivations over the rules in PLL$_2$ is also denoted by PLL$_2$. The *propositional* fragment of PLL$_2$ (both the set of rules and the set of its derivations) is denoted by PLL.

We set nuPLL$_2$:= $\{$ax, cut, $\otimes$, $\mathcal{B}$, 1, $\perp$, ?b, ?w, ib!p, $\forall$, $\exists\}$ and the set of derivations over the rules in nuPLL$_2$ is also noted nuPLL$_2$.[4]

Akin to light linear logics [30, 35, 44], the exponential rules of PLL$_2$ are weaker than those in MELL$_2$: the usual promotion rule is replaced by f!p (*functorial promotion*), and the usual contraction and dereliction rules by ?b. As a consequence, the *digging* formula $!A \multimap$ $!!A$ and the *contraction* formula $!A \multimap !A \otimes !A$ are not provable in PLL$_2$ (unlike the dereliction formula, Example 8). It is easy to show that MELL$_2$ = PLL$_2 \cup \{$??d $\}$: if we add the digging formula as an axiom (or equivalently, the *digging rule* ??d in Figure 2) to PLL$_2$, then the contraction formula becomes provable, and the expressivity of the obtained proof system coincides with MELL$_2$.

The system nuPLL$_2$ is our formulation of Mazza and Terui's nuPL$_{\forall \ell}$ [40], but unlike the latter, the *co-absorbtion rule* ?b (Figure 2 on the right) is not in nuPLL$_2$. Despite it, we shall prove that nuPLL$_2$ still captures **FP**/poly.

Example 8. Figure 3 gives some examples of derivation in PLL$_2$. The (distinct) derivations $\underline{0}$ and $\underline{1}$ prove the same formula $\mathbf{B} = \forall X. (X^{\perp} \,\mathcal{B}\, X^{\perp}) \,\mathcal{B}\, (X \otimes X)$, where X_1, X_2, X_3, X_4 are distinct occurrences of the variable X. Derivations $\mathcal{D}_{\text{abs}}$ and $\mathcal{D}_{\text{der}}$ respectively prove the *absorption law* $!A \multimap A \otimes !A$ and the *dereliction law* $!A \multimap A$.

---

[3]The (!, ?)-freeness of the formula instantiated in the existential rule ($\exists$) is crucial for establishing a polynomial bound on cut-elimination. This linearity restriction prevents the encoding of exponential functions (see Remark 92 in Appendix D.5).

[4]This requires a slight change in Definition 1: the tree labelled by a derivation in nuPLL$_2$ must be over $\mathbb{N}^{\omega}$ instead of $\{1, 2\}^*$, to deal with infinitely branching derivations. Basically, nuPLL$_2$ is obtained from PLL$_2$ by replacing the rule f!p with ib!p.

$$\text{ax} \frac{}{A, A^\perp} \qquad \text{cut} \frac{\Gamma, A \quad A^\perp, \Delta}{\Gamma, \Delta} \qquad \Re \frac{\Gamma, A, B}{\Gamma, A \Re B} \qquad \otimes \frac{\Gamma, A \quad B, \Delta}{\Gamma, \Delta, A \otimes B} \qquad 1 \frac{}{1} \qquad \perp \frac{\Gamma}{\Gamma, \perp} \quad \Bigg| \quad \text{ib!p} \frac{\overset{\mathcal{D}_0}{\Gamma, A} \quad \cdots \quad \overset{\mathcal{D}_n}{\Gamma, A} \quad \cdots}{?\Gamma, !A} \ \{\mathcal{D}_i \mid i \in \mathbb{N}\} \text{ is finite}$$

$$\text{f!p} \frac{\Gamma, A}{?\Gamma, !A} \qquad \text{?w} \frac{\Gamma}{\Gamma, ?A} \qquad \text{?b} \frac{\Gamma, A, ?A}{\Gamma, ?A} \qquad \forall \frac{\Gamma, A}{\Gamma, \forall X.A} X \notin FV(\Gamma) \qquad \exists \frac{\Gamma, A[B/X]}{\Gamma, \exists X.A} B \text{ is (!,?)-free} \quad \Bigg| \quad \text{!w} \frac{}{!A} \qquad \text{!b} \frac{\Gamma, A \quad \Delta, !A}{\Gamma, \Delta, !A} \qquad \text{??d} \frac{\Gamma, ??A}{\Gamma, ?A} \qquad \text{c!p} \frac{\Gamma, A \quad ?\Gamma, !A}{?\Gamma, !A}$$

Figure 2: Sequent calculus rules of PLL_2 **(on the right) and other rules we consider in this paper (on the left)**

Figure 3: Examples of derivations in PLL_2 **(**$\underline{1}, \underline{0}, \mathcal{D}_{\mathsf{abs}}, \mathcal{D}_{\mathsf{der}}$**, on the left) and of coderivations in** PLL_2^∞ **(**$\mathcal{D}_\perp, \mathcal{D}_?$**, on the right).**

The **cut-elimination** relation $\rightarrow_{\mathsf{cut}}$ in PLL_2 is the union of cut-elimination steps in Figure 4 (the non-commutative steps are called *principal*). The reflexive-transitive closure of $\rightarrow_{\mathsf{cut}}$ is noted $\rightarrow_{\mathsf{cut}}^*$.

Termination of cut-elimination in PLL has been proved in [2], and extends straightforwardly to PLL_2.

THEOREM 9 ([2]). *For every* $\mathcal{D} \in \mathsf{PLL}_2$*, there is a cut-free* $\mathcal{D}' \in \mathsf{PLL}_2$ *such that* $\mathcal{D} \rightarrow_{\mathsf{cut}}^* \mathcal{D}'$.

A byproduct of our grand tour diagram in Figure 1 is that PLL_2 represents exactly the class of functions in **FP**. To see this, we introduce a rather permissive notion of representability for PLL_2, along the lines of [28]. This notion smoothly adapts to other proof systems we shall study in this paper.

Definition 10 (Representability). A set T is **represented in** PLL_2 by a formula $\mathbf{T}$ if there is an injection $(\underline{\cdot})$ from T to the set of cut-free derivations in PLL_2 with conclusion $\mathbf{T}$.

A derivation $\mathcal{D}$ in PLL_2 **represents** a (total) function $f: T_1 \times \ldots \times T_n \rightarrow T$ if it proves $\mathbf{T_1} \multimap \ldots \multimap \mathbf{T_n} \multimap \mathbf{T}$ where $\mathbf{T_1}, \ldots, \mathbf{T_n}, \mathbf{T}$ represent $T_1, \ldots, T_n, T$ respectively, and for all $x_1 \in T_1, \ldots, x_n \in T_n$, the reduction in Figure 5 holds. A (total) function $f: T_1 \times \ldots \times T_n \rightarrow T$ is **representable in** PLL_2 if there is a derivation in PLL_2 representing f. We denote by $\underline{f}$ a derivation representing f.

Example 11. The set of Booleans $\mathbb{B} = \{\mathbf{0}, \mathbf{1}\}$ is represented in PLL_2 by the formula $\mathbf{B}$ in Example 8 thanks to the derivations $\underline{0}$ and $\underline{1}$ in Figure 3. The set $\{\mathbf{0}, \mathbf{1}\}^*$ of Boolean strings is represented by the formula $\mathbf{S} := \forall X.!(\mathbf{B} \multimap X \multimap X) \multimap X \multimap X$. We will actually mainly work with a parametric version of $\mathbf{S}$, i.e., the instantiation $\mathbf{S}[A] := !(\mathbf{B} \multimap A \multimap A) \multimap A \multimap A$ for any formula A. We write $\mathbf{S}[]$ to denote $\mathbf{S}[A]$ for some A. Each string $b_1 \cdots b_n \in \{\mathbf{0}, \mathbf{1}\}^*$ is then encoded in PLL_2 by the derivation $\underline{b_1 \cdots b_n}$ of $\mathbf{S}[A]$ below.

4 NON-WELLFOUNDED SECOND ORDER PARSIMONIOUS LINEAR LOGIC

4.1 From infinite branching to non-wellfounded

In this paper we explore a dual approach to the one developed in [40]: instead of considering (wellfounded) derivations with infinite branching, like nuPLL_2, we introduce (non-wellfounded) coderivations with finite branching. For this purpose, the infinitary rule ib!p of nuPLL_2 is replaced by the binary rule called **conditional promotion** (c!p) from Figure 2, and we define PLL_2^∞ as the set of coderivations generated by the same rules as PLL_2, except that f!p is replaced by c!p.

Definition 12. We set $\mathsf{PLL}_2^\infty := \{\mathsf{ax}, \otimes, \Re, 1, \perp, \mathsf{cut}, ?\mathsf{b}, ?\mathsf{w}, \mathsf{c!p}, \forall, \exists\}$. The set of coderivations over the rules in PLL_2^∞ is also noted PLL_2^∞.

From now on, we only consider coderivations in PLL_2^∞.

We can embed PLL_2 and nuPLL_2 into PLL_2^∞ via the conclusion-preserving translations $(\cdot)^\circ$ and $(\cdot)^\bullet$ defined in Figure 8. These translations expand the promotion rules f!p and ib!p into non-wellfounded coderivations as in Figure 7 defined as follows.

Definition 13. A **non-wellfounded box** (nwb for short) is a coderivation $\mathfrak{S} \in \mathsf{PLL}_2^\infty$ with an infinite branch $\{\epsilon, 2, 22, \ldots\}$ (called the **main branch**) made of conclusions of c!p-rules (see Figure 7).

We write $\mathfrak{S} = \mathsf{c!p}_{(\mathcal{D}_0, \ldots, \mathcal{D}_n, \ldots)}$ if the sub-coderivation rooted in $v_i = 1^{i+1}$ (i.e., the word over $\{1\}$ of length $i + 1$) is $\mathcal{D}_i = \mathfrak{S}_{v_i}$ (also

Figure 4: Cut-elimination steps in PLL_2**.**

Figure 5: Representability of a function $f : T_1 \times \ldots \times T_n \to T$**.**

Figure 6: Exponential cut-elimination steps in nuPLL_2**.**

Figure 7: A non-wellfounded box in PLL_2^∞**.**

noted $\mathfrak{S}(i)$ and called the ith **call** of $\mathfrak{S}$. The **principal formula** of the nwb is the unique !-formula in the conclusion of $\mathfrak{S}$.

Let $\mathrm{Calls}(\mathfrak{S}) = \{\mathfrak{S}(i) \mid i \in \mathbb{N}\}$ be the set of **calls** of $\mathfrak{S}$. We say that $\mathfrak{S}$ has **finite support** if $\mathrm{Calls}(\mathfrak{S})$ is finite. A coderivation $\mathcal{D}$ has **finite support** if any nwb in $\mathcal{D}$ is so.

Example 14. A nwb $\mathfrak{S} = \mathsf{c!p}_{(\mathcal{D}_0,\ldots,\mathcal{D}_n,\ldots)}$ as in Figure 7 with conclusion !B and $\mathcal{D}_i \in \{\underline{\mathbf{0}}, \underline{\mathbf{1}}\}$ for each $i \in \mathbb{N}$ has finite support.

The reader familiar with linear logic can see a nwb as a box with possibly *infinitely* many distinct contents (its calls), in contrast with regular boxes (identified with f!p-rules) that can only provide infinitely many copies of the *same* content.

The *cut-elimination* steps $\to_{\mathrm{cut}}$ for PLL_2^∞ are in Figure 4 (except for exponentials) and Figure 9. Computationally, we interpret the step c!p-vs-?b as the pop operator allowing us to access the head of

Figure 8: Bottom-up translations $(\cdot)^\circ \colon \mathsf{PLL}_2 \to \mathsf{PLL}_2^\infty$, **and** $(\cdot)^\bullet \colon \mathsf{nuPLL}_2 \to \mathsf{PLL}_2^\infty$. **Rules in** $\mathsf{PLL}_2 \setminus \{\mathsf{f!p}\}$ **are translated to themselves.**

a stream encoded by a nwb. Note that, unlike the streams encoded using the rule ib!p, streams encoded by nwbs may have infinitely many distinct elements (one for each call).

The notion of representability for PLL_2^∞ can be obtained by adapting Definition 10 to coderivations in PLL_2^∞.

4.2 Totality via a progressing criterion

A non-wellfounded system such as PLL_2^∞ is inconsistent. Indeed, the non-wellfounded coderivation $\mathcal{D}_\perp$ in Figure 3 (on the right) shows that any non-empty sequent is provable in PLL_2^∞.

The coderivation $\mathcal{D}_\perp$ in Figure 3 is not cut-free, and if $\mathcal{D}_\perp \to_{\mathsf{cut}} \mathcal{D}$ then $\mathcal{D} = \mathcal{D}_\perp$. Thus $\mathcal{D}_\perp$ cannot reduce to a cut-free coderivation, and the cut-elimination theorem cannot hold in PLL_2^∞. From a computational point of view, this means that the proof system PLL_2^∞ can represent non-total functions.

In non-wellfounded proof theory, the usual way to recover logical consistency, and so—computationally—*totality* of representable functions, is to introduce a global soundness condition on coderivations, the *progressing criterion* [18, 20, 33]. In PLL_2^∞, this criterion relies on tracking occurrences of !-formulas in coderivations [2].

Definition 15. Let $\mathcal{D}$ be a coderivation in PLL_2^∞. An occurrence of a formula in a premise of a rule r is the **parent** of an occurrence of a formula in the conclusion if they are connected according to the edges depicted in Figure 10. A !-**thread** (resp. ?-**thread**) in $\mathcal{D}$ is a maximal sequence $(A_i)_{i \in I}$ of !-formulas (resp. ?-formulas) for some downward-closed $I \subseteq \mathbb{N}$ such that A_{i+1} is the parent of A_i for all $i \in I$. A !-thread $(A_i)_{i \in I}$ is **progressing** if A_j is in the conclusion of a c!p for infinitely many $j \in I$. $\mathcal{D}$ is **progressing** if every infinite branch contains a progressing !-thread. We define pPLL_2^∞ as the set of progressing coderivations in PLL_2^∞.

Example 16. Coderivations in Figure 3 are not progressing: the rightmost branch of $\mathcal{D}_\perp$, i.e., the branch $\{\epsilon, 2, 22, \ldots\}$, and the unique branch of $\mathcal{D}_?$ are infinite and contain no c!p-rules. By contrast, the nwb $\mathsf{c!p}_{(i_0, \ldots, i_n, \ldots)}$ discussed in Example 14 is progressing since the only infinite branch is its main branch, which contains a !-thread of formulas !A, each one principal for a c!p rule.

The regular coderivation below is not progressing: the branch $\{\epsilon, 2, 21, 212, 2121, \ldots\}$ is infinite but has no progressing !-thread.

Remark 17. Any infinite branch in a progressing coderivation $\mathcal{D}$ contains exactly one progressing !-thread. This follows from

maximality of !-threads and the fact that conclusions of c!p-rules contain at most one !-formula. As a consequence, any infinite !-thread in a branch of $\mathcal{D}$ must be progressing.

Theorem 18 (Continuous cut-elimination [2]). *For each $\mathcal{D} \in \mathsf{pPLL}_2^\infty$, there is a cut-free $\mathcal{D}' \in \mathsf{pPLL}_2^\infty$ with the same conclusion.*

This result has been proved in [2] for the *propositional* PLL_2^∞ (i.e., without second-order). The proof smoothly extends to the whole PLL_2^∞. Indeed, thanks to (!, ?)-freeness for the formula instantiated in rule $\exists$, the cut-elimination step $\exists$-vs-$\forall$ does not change the *geometry* of the derivation, and !-threads never end in an instantiation.

4.3 Recovering (weak forms of) regularity

The progressing criterion cannot capture the finiteness condition in the rule ib!p of nuPLL_2, as the following example shows:

Example 19. Consider the nwb $\mathfrak{S} = \mathsf{c!p}_{(\mathcal{D}_0, \ldots, \mathcal{D}_n, \ldots)}$ as in Figure 7 with conclusion !!$\mathbf{B}$ and with $\mathcal{D}_i = \mathsf{c!p}_{(\underline{1}, \ldots, \underline{1}, \underline{0}, \ldots)}$. The nwb is progressing but is not in the image of a rule ib!p via $(\cdot)^\bullet$ (see Figure 8) as it has infinitely many distinct calls.

To identify in pPLL_2^∞ the coderivations corresponding to derivations in nuPLL_2 and in PLL_2 via the translations $(\cdot)^\bullet$ and $(\cdot)^\circ$, respectively, we need additional conditions.

Definition 20. A coderivation is **weakly regular** if it has only finitely many distinct sub-coderivations whose conclusions are left premises of c!p-rules; it is **finitely expandable** if any branch contains finitely many cut and ?b rules. We denote by wrPLL_2^∞ (resp. rPLL_2^∞) the set of weakly regular (resp. regular) and finitely expandable coderivations in pPLL_2^∞.

Remark 21. Regularity implies weak regularity and the converse fails, see Example 22 below, so $\mathsf{rPLL}_2^\infty \subsetneq \mathsf{wrPLL}_2^\infty$. A progressing and finitely expandable $\mathcal{D} \in \mathsf{PLL}_2^\infty$ is weakly regular if and only if any nwb in $\mathcal{D}$ has finite support.

Example 22. $\mathcal{D}_\perp$ and $\mathcal{D}_?$ in Figure 3 (on the right) are weakly regular (they have no c!p rules) but not finitely expandable (their infinite branch has infinitely many cut or ?b). Coderivation in Example 19 is not weakly regular (it has infinitely many distinct calls).

An example of a weakly regular but not regular coderivation is the nwb $\mathsf{c!p}_{(i_0, \ldots, i_n, \ldots)}$ in Example 14 when the infinite sequence $(i_j)_{j \in \mathbb{N}} \in \{\mathbf{0}, \mathbf{1}\}^\omega$ is not periodic: $\underline{0}$ and $\underline{1}$ are the only coderivations ending in the left premise of a c!p rule (so the nwb is weakly regular), but there are infinitely many distinct coderivations ending in the right premise of a c!p rule (so the nwb is not regular). Moreover, that nwb is finitely expandable, as it contains no ?b or cut.

By inspecting Figures 4 and 9 for PLL_2^∞, we prove the following.

Figure 9: Exponential cut-elimination steps for coderivations of PLL_2^∞.

Figure 10: PLL_2^∞ rules: edges connect a formula in the conclusion with its parent(s) in a premise.

Proposition 23. *Cut elimination preserves weak-regularity, regularity and finite expandability. Therefore, if $\mathcal{D} \in \mathsf{X}$ with $\mathsf{X} \in \{\mathsf{rPLL}_2^\infty, \mathsf{wrPLL}_2^\infty\}$ and $\mathcal{D} \to_{\mathsf{cut}} \mathcal{D}'$, then also $\mathcal{D}' \in \mathsf{X}$.*

By an argument similar to [12, Corollary 32] we have that it is **NL**-decidable if a regular coderivation is in rPLL_2^∞. Of course a similar decidability result cannot hold for wrPLL_2^∞, this proof system containing continuously many coderivations, as hinted by the nwb depicted in Example 14.

4.4 Simulation results

The translations $(\cdot)^\circ$ and $(\cdot)^\bullet$ in Figure 8 map the rules f!p and ib!p to infinite sequences of rules. So, the cut-elimination step f!p-vs-f!p in PLL_2 (resp. ib!p-vs-ib!p in nuPLL_2) can only be simulated by infinitely many cut-elimination steps in rPLL_2^∞ (resp., wrPLL_2^∞). However, as long as we consider representability of functions on Boolean strings, we can show that those cut-elimination steps are not needed to reduce a derivation of PLL_2 (resp. nuPLL_2) to a cut-free one. As a consequence, computations over binary strings in PLL_2 (resp. nuPLL_2) can be simulated in rPLL_2^∞ (resp. wrPLL_2^∞) using within a *finite* number of cut-elimination steps.

Theorem 24. *[Simulation] Let $f : (\{0,1\}^*)^n \to \{0,1\}^*$.*

(1) If f is representable in nuPLL_2, then so it is in wrPLL_2^∞.
(2) If f is representable in PLL_2, then so it is in rPLL_2^∞.

PROOF SKETCH. We only prove Item 1, as 2 is proven similarly. Let $\mathcal{D} : \mathsf{S}[] \multimap \overset{n \geq 0}{\cdots} \multimap \mathsf{S}[] \multimap \mathsf{S}$ represent $f : (\{0,1\}^*)^n \to \{0,1\}^*$ in nuPLL_2 and let $x_1, \ldots, x_n \in \{0,1\}^*$. This means that the reduction in Figure 5 holds for $\mathsf{T}_1 = \ldots = \mathsf{T}_n = \mathsf{S}[]$ and $\mathsf{T} = \mathsf{S}$. Let $\sigma := \mathcal{D}_0 \to_{\mathsf{cut}} \mathcal{D}_1 \to_{\mathsf{cut}} \ldots \to_{\mathsf{cut}} \mathcal{D}_n = \underline{f(s_1, \ldots, s_n)}$ be such a reduction sequence. It is easy to see that, if $\mathcal{D} \in \mathsf{nuPLL}_2$ has !-free conclusion and is not cut-free, then $\mathcal{D}$ contains a cut $r \notin \{\mathsf{f!p\text{-}vs\text{-}f!p}, \mathsf{ib!p\text{-}vs\text{-}ib!p}\}$. Since each $\mathcal{D}_i$ has S as a conclusion, which is !-free, σ can be rewritten to another cut-elimination sequence $\widehat{\sigma}$ that never applies the steps f!p-vs-f!p and ib!p-vs-ib!p. We conclude by noticing that, if $\mathcal{D}_1 \to_{\mathsf{cut}} \mathcal{D}_2$ does not reduce a cut $r \notin \{\mathsf{f!p\text{-}vs\text{-}f!p}, \mathsf{ib!p\text{-}vs\text{-}ib!p}\}$, then $\mathcal{D}^\bullet \to_{\mathsf{cut}} \mathcal{D}_2^\bullet$ (see Figure 8), and by observing that $\underline{s}^\bullet = \underline{s}$ for any binary string $s \in \{0,1\}^*$. □

4.5 Approximating coderivations

In this subsection we introduce *open coderivations*, which approximate coderivations, and show a decomposition property for finitely expandable and progressing coderivations.

Definition 25. We define the set of rules $\mathsf{oPLL}_2^\infty := \mathsf{PLL}_2^\infty \cup \{\mathsf{hyp}\}$, where $\mathsf{hyp} := \mathsf{hyp} \dfrac{}{\Gamma}$ for any sequent Γ.[5] We will also refer to oPLL_2^∞ as the set of coderivations over oPLL_2^∞, which we call **open coderivations**. An **open derivation** is a derivation in oPLL_2^∞.

Definition 26. Let $\mathcal{D}$ be an open coderivation and $\mathcal{V} = \{v_1, \ldots, v_n\} \subseteq \{1,2\}^*$ be a finite set of mutually incomparable nodes of $\mathcal{D}$ (w.r.t. the prefix order). If $\{\mathcal{D}_i'\}_{1 \leq i \leq n}$ is a set of open coderivations where $\mathcal{D}_i'$ has the same conclusion as the subderivation $\mathcal{D}_{v_i}$ of $\mathcal{D}$, denote by $\mathcal{D}(\mathcal{D}_1'/v_1, \ldots, \mathcal{D}_n'/v_n)$, the open coderivation obtained by replacing each $\mathcal{D}_{v_i}$ with $\mathcal{D}_i'$. The **pruning** of $\mathcal{D}$ over $\mathcal{V}$ is the open coderivation $\|\mathcal{D}\|_{\mathcal{V}} = \mathcal{D}(\mathsf{hyp}/v_1, \ldots, \mathsf{hyp}/v_n)$.

If $\mathcal{D}$ and $\mathcal{D}'$ are open coderivations, we say that $\mathcal{D}$ is an **approximation** of $\mathcal{D}'$ (noted $\mathcal{D} \leq \mathcal{D}'$) iff $\mathcal{D} = \|\mathcal{D}'\|_{\mathcal{V}}$ for some $\mathcal{V} \subseteq \{1,2\}^*$. An approximation is **finite** if it is an open derivation.

Cut-elimination steps do not increase the size of open derivations:

Proposition 27 (Cubic bound). *Let $\mathcal{D}$ be an open derivation and let $\mathsf{S}(\mathcal{D})$ be the maximum number of $?$-formulas in the conclusion of a c!p rule of $\mathcal{D}$. If $\mathcal{D} = \mathcal{D}_0 \to_{\mathsf{cut}} \cdots \to_{\mathsf{cut}} \mathcal{D}_n$ then:*

(1) n and $|\mathcal{D}_i|$ are in $O(\mathsf{S}(\mathcal{D})^3 \cdot |\mathcal{D}|^3)$ for any $i \in \{0, \ldots, n\}$.
(2) If only principal cut-elimination steps are applied then n and $|\mathcal{D}_i|$ are in $O(\mathsf{S}(\mathcal{D}) \cdot |\mathcal{D}|)$ for any $i \in \{0, \ldots, n\}$.
(3) If the reduction sequence is maximal then $\mathcal{D}_n$ is cut free.

PROOF SKETCH. The only cut-elimination step that may increase the number of rules of a derivation $\mathcal{D}$ is c!p-vs-?b, which introduces at most $\mathsf{S}(\mathcal{D})$ new ?b rules. So, any cut-elimination sequence starting from $\mathcal{D}$ has at most $O(\mathsf{S}(\mathcal{D}) \cdot |\mathcal{D}|)$ principal steps, alternated by sequences of commutative steps (each one having at most a quadratic number of steps in the size of the derivation). □

Progressing and finitely expandable coderivations can be represented and approximated in a canonical way:

---

[5]Previously introduced notions and definitions on coderivations extend to open coderivations in the obvious way, e.g. the global conditions Definition 15 and Definition 20, as well as the cut-elimination relation $\to_{\mathsf{cut}}$.

Proposition 28. *Let $\mathcal{D} \in \mathrm{pPLL}_2^\infty$ be finitely expandable. There is a prebar $\mathcal{V} \subseteq \{1, 2\}^*$ of $\mathcal{D}$ such that each $v \in \mathcal{V}$ is the root of a nwb.*

Definition 29. Let $\mathcal{D} \in \mathrm{pPLL}_2^\infty$ be finitely expandable. The **decomposition prebar** of $\mathcal{D}$ is the minimal prebar $\mathcal{V}$ of $\mathcal{D}$ such that, for all $v \in \mathcal{V}$, $\mathcal{D}_v$ is a nwb. We denote with $\mathrm{border}(\mathcal{D})$ such a prebar and we set $\mathrm{base}(\mathcal{D}) \coloneqq \lfloor\!\lfloor \mathcal{D} \rfloor\!\rfloor_{\mathrm{border}(\mathcal{D})}$.

By weak König lemma, $\mathrm{border}(\mathcal{D})$ is finite and $\mathrm{base}(\mathcal{D})$ is a finite approximation of $\mathcal{D}$.

5 SOUNDNESS

In this section we establish the soundness theorem, i.e., every function over binary strings that is representable in wrPLL_2^∞ (resp. rPLL_2^∞) is in $\mathbf{FP}/\mathrm{poly}$ (resp. $\mathbf{FP}$). The key result is a polynomial modulus of continuity for cut-elimination (Lemma 50), from which we can extract a family of polynomial size circuits C_f computing f. We conclude by observing that C_f is, in fact, uniform whenever the coderivation representing f is regular.

5.1 Nesting, depth, cosize and truncation

Akin to linear logic, the *depth* of a coderivation is the maximal number of nested nwbs.

Definition 30. Let $\mathcal{D} \in \mathrm{PLL}_2^\infty$. The **nesting level of a sequent occurrence** Γ in $\mathcal{D}$ is the number $\mathbf{nl}_{\mathcal{D}}(\Gamma)$ of nodes below it that are the root of a call of a nwb. The **nesting level of a formula (occurrence)** A in $\mathcal{D}$, noted $\mathbf{nl}_{\mathcal{D}}(A)$, is the nesting level of the sequent that contain that formula. The **nesting level of a rule** r in $\mathcal{D}$, noted $\mathbf{nl}_{\mathcal{D}}(\mathsf{r})$ (resp. **of a sub-coderivation** $\mathcal{D}'$ of $\mathcal{D}$, noted $\mathbf{nl}_{\mathcal{D}}(\mathcal{D}')$), is the nesting level of the conclusion of r (resp. conclusion of $\mathcal{D}'$).

The **depth of** $\mathcal{D}$ is $\mathbf{d}(\mathcal{D}) \coloneqq \sup_{\mathsf{r} \in \mathcal{D}}\{\mathbf{nl}_{\mathcal{D}}(\mathsf{r})\} \in \mathbb{N} \cup \{\infty\}$.

Remark 31. Let $\mathfrak{S}$ be a nwb with $\mathbf{nl}_{\mathcal{D}}(\mathfrak{S}) = n$. If A is a formula occurrences in the main branch of $\mathfrak{S}$ then $\mathbf{nl}_{\mathcal{D}}(A) = n$, and $\mathbf{nl}_{\mathcal{D}}(\mathfrak{S}(i)) = n + 1$ for every $i \geq 0$.

Proposition 32 ([2])**.** *If $\mathcal{D}$ is weakly regular then $\mathbf{d}(\mathcal{D}) \in \mathbb{N}$. Moreover, $\mathcal{D} \rightarrow_{\mathrm{cut}} \mathcal{D}'$ implies $\mathbf{d}(\mathcal{D}) \geq \mathbf{d}(\mathcal{D}')$.*

We introduce a notion of (finite) size for coderivations in wrPLL_2^∞, called *cosize*, relying on Proposition 32.

Definition 33. Let $\mathcal{D} \in \mathrm{wrPLL}_2^\infty$, and $\mathrm{border}(\mathcal{D}) = \{v_1, \dots, v_k\}$ be its decomposition prebar (thus $\mathfrak{S}_i \coloneqq \mathcal{D}_{v_i}$ is a nwb for all $1 \leq i \leq k$). We define the **cosize of** $\mathcal{D}$, written $||\mathcal{D}||$, by induction on $\mathbf{d}(\mathcal{D})$. If $\mathbf{d}(\mathcal{D}) = 0$ then $\mathcal{D} = \mathrm{base}(\mathcal{D})$ and we set $||\mathcal{D}|| \coloneqq |\mathcal{D}|$. Otherwise $\mathbf{d}(\mathcal{D}) > 0$, and $||\mathcal{D}|| \coloneqq |\mathrm{base}(\mathcal{D})| + \sum_{i=1}^{k} \sum_{\mathcal{D}' \in \mathrm{Calls}(\mathfrak{S}_i)} ||\mathcal{D}'||$. Notice that, by Remark 21, $\mathrm{Calls}(\mathfrak{S}_i)$ is a *finite* set for any $i \in \{1, \dots, k\}$. The **cosize at depth** d, written $||\mathcal{D}||_d$, is defined for all $d \leq \mathbf{d}(\mathcal{D})$ as $||\mathcal{D}||_0 = |\mathrm{base}(\mathcal{D})|$, and as $||\mathcal{D}||_{d+1} = \max\{||\mathcal{D}'||_d \mid \mathcal{D}' \in \mathrm{Calls}(\mathfrak{S}_i)$ for some $1 \leq i \leq k\}$.

An *n-(hyper)truncation* is a particular finite approximation for coderivations that only considers the first n calls of each nwb.

Definition 34. A **finite non-wellfounded promotion** is defined as a coderivation $\mathfrak{F} = \mathsf{c!p}_{\langle \mathcal{D}_0, \mathcal{D}_1, \dots, \mathcal{D}_{n-1}\rangle}$ in Figure 11. We may write $\mathfrak{F}(i)$ to denote $\mathcal{D}_i$.

Let $\mathcal{D} \in \mathrm{wrPLL}_2^\infty$ with $\mathrm{border}(\mathcal{D}) = \{v_1, \dots, v_k\}$ (so $\mathfrak{S}_i \coloneqq \mathcal{D}_{v_i}$ is a nwb for all $1 \leq i \leq k$). The *n-truncation* $\lfloor\!\lfloor \mathcal{D} \rfloor\!\rfloor_n$ and the *n-hypertruncation* $\lfloor \mathcal{D} \rfloor_n$ of $\mathcal{D}$ are the open derivations defined

Figure 11: A finite non-wellfounded promotion.

for all $n > 0$ as follows: if $\mathbf{d}(\mathcal{D}) = 0$, then $\lfloor\!\lfloor \mathcal{D} \rfloor\!\rfloor_n = \lfloor \mathcal{D} \rfloor_n \coloneqq \mathrm{base}(\mathcal{D}) = \mathcal{D}$, and if $\mathbf{d}(\mathcal{D}) > 0$, then

$$\lfloor\!\lfloor \mathcal{D} \rfloor\!\rfloor_n \coloneqq \mathrm{base}(\mathcal{D})(\vec{\mathfrak{F}_i}/\vec{v_i}) \qquad \lfloor \mathcal{D} \rfloor_n \coloneqq \mathrm{base}(\mathcal{D})(\vec{\mathfrak{F}_i'}/\vec{v_i})$$

where for all $i \in \{1, \dots, k\}$, $\mathfrak{F}_i = \mathsf{c!p}_{\langle \lfloor\!\lfloor \mathfrak{S}_i(0)\rfloor\!\rfloor_n, \dots, \lfloor\!\lfloor \mathfrak{S}_i(n-1)\rfloor\!\rfloor_n\rangle}$ and $\mathfrak{F}_i' = \mathsf{c!p}_{\langle \mathrm{base}(\mathfrak{S}_i(0)), \dots, \mathrm{base}(\mathfrak{S}_i(n-1))\rangle}$.

Notice that $\lfloor\!\lfloor \mathcal{D} \rfloor\!\rfloor_n$ and $\lfloor \mathcal{D} \rfloor_n$ are finite, and $\lfloor \mathcal{D} \rfloor_n \preceq \lfloor\!\lfloor \mathcal{D} \rfloor\!\rfloor_n$.

5.2 Exponential flows

In this subsection we introduce the *exponential graph* of a coderivation $\mathcal{D}$, a directed graph associated to $\mathcal{D}$ that allows us a static analysis of the exponential cut-elimination steps performed at depth 0. Directed paths in this graph, called *exponential flows*, can be then used to precompute the maximum number of calls of a nwb that decrease their nesting level by reducing a cut $\mathsf{c!p}$-vs-$?\mathsf{b}$. This number will be called *rank* of $\mathcal{D}$, and plays a crucial role for establishing a polynomial bound on cut-elimination.

Definition 35 (Exponential flow)**.** Let $\mathcal{D} \in \mathrm{wrPLL}_2^\infty$. The **exponential graph of** $\mathcal{D}$, written $\mathcal{G}(\mathcal{D})$, is a directed acyclic graph whose nodes are (labelled by) the $?$-formulas and the $!$-formulas with nesting level 0 that occur in $\mathcal{D}$, and whose directed edges connect a node A to a node B if:

- $A = ?C^\perp$ and $B = !C$ are the conclusions of an ax rule;
- $A = ?C^\perp$ and $B = !C$ are conclusions of a $\mathsf{c!p}$ rule;
- $A = ?C$ is principal for a $?\mathsf{b}$-rule with active formula $B = ?C$;
- $A = !C$ and $B = ?C^\perp$ are the cut-formulas of a cut rule;
- A and B are $?$-formulas (resp. $!$-formulas) in a context and B (resp. A) is an immediate ancestor of A (resp. B).

see, e.g., Example 36 and Figure 12. A $!$-**node** (resp., $?$-**node**) is a node labelled by a $!$-formula (resp., $?$-formula). A b-**node** (resp., w-**node**) is a node labelled by the principal formula for a $?\mathsf{b}$ rule (resp., for a $?\mathsf{w}$ rule). A p-**node** (resp., a-**node**) is a node labelled by the principal $!$-formula (resp., principal $?$-formula) for a $\mathsf{c!p}$ rule *in the main branch of a nwb*.

If ϕ is a directed path of $\mathcal{G}(\mathcal{D})$ crossing a node A, then $\phi[A]$ is the maximal directed subpath of ϕ that starts at A. We also say that ϕ **crosses** a cut rule (resp. a nwb) when it crosses its active formulas (resp., when it crosses its principal $!$-formula and one of its principal $?$-formulas). An **exponential flow** is a maximal directed path of $\mathcal{G}(\mathcal{D})$ that does not cross infinitely p- and a-nodes.

Example 36. Let $\mathcal{D}$ be the following nwb with conclusion $?A^\perp, !A$ and whose calls are axioms:

$$\text{(1)}$$

The nodes of $\mathcal{G}(\mathcal{D})$ are the formula occurrences $?A^\perp$, $!A$ in the main branch of $\mathcal{D}$, as they all have nesting level 0 by Remark 31. The exponential flows of $\mathcal{D}$ are all *finite* directed paths from the bottommost occurrence of $?A^\perp$ to the bottommost occurrence of $!A$.

We introduce some useful measures on exponential flows, e.g. *rank*. We then show some basic properties of the exponential flows.

Definition 37 (Measures). Let $\mathcal{D} \in \text{wrPLL}_2^\infty$, and let ϕ be a directed path in $\mathcal{G}(\mathcal{D})$. If $\mathfrak{S}$ is a nwb with principal !-formula $!A$, we define $\mathsf{p}(\phi, \mathfrak{S})$ as the number of p-nodes of the nwb $\mathfrak{S}$ crossed by ϕ.

The **rank of** ϕ, written $\mathsf{b}(\phi)$, is the number of b-nodes ϕ crosses. The **rank of** $\mathcal{D}$, written $\mathsf{rk}(\mathcal{D})$, is the number b-nodes of $\mathcal{G}(\mathcal{D})$.

Remark 38. If $\mathcal{D} \in \text{wrPLL}_2^\infty$ then any exponential flow ϕ of $\mathcal{G}(\mathcal{D})$ is finite and $\mathsf{b}(\phi) \leq \mathsf{rk}(\mathcal{D}) \in \mathbb{N}$.

Residues allow us to track exponential flows during cut-elimination.

Definition 39 (Residue). Let $\mathcal{D} \in \text{wrPLL}_2^\infty$, and let ϕ be an exponential flow of $\mathcal{D}$. A **residue of** ϕ **along** $\mathcal{D} \rightarrow_{\text{cut}}^* \mathcal{D}'$ is an exponential flow $\widehat{\phi}$ of $\mathcal{D}'$ such that all nodes $A \in \mathcal{G}(\mathcal{D}) \cap \mathcal{G}(\mathcal{D}')$ crossed by $\widehat{\phi}$ are also crossed by ϕ.

The following example shows that the rank of a residue of an exponential flow ϕ can have rank greater than the rank of ϕ.

Example 40. Suppose $\mathcal{D} \rightarrow_{\text{cut}} \mathcal{D}'$ be the following multiplicative cut-elimination step reducing a cut with active formulas $A \,\mathcal{S}\, B$ and $A^\perp \otimes B^\perp$, where A is a !-node of $\mathcal{G}(\mathcal{D})$ and $A^\perp$ is a ?-node of $\mathcal{G}(\mathcal{D})$:

Let ϕ be and ϕ' be exponential flows of $\mathcal{G}(\mathcal{D})$ crossing A and $A^\perp$, respectively. It is easy to see that no node of $\mathcal{G}(\mathcal{D})$ is crossed by both ϕ and ϕ'. However, ϕ and ϕ' have the same residue in $\mathcal{D}'$, which is the exponential flow $\widehat{\phi} = \phi b \phi'$ such that $\mathsf{b}(\widehat{\phi}) = \mathsf{b}(\phi) + \mathsf{b}(\phi')$.

Despite the above example, as long as we consider cut-elimination steps reducing exponential cuts, residues of an exponential flow cannot have greater rank.

Proposition 41. *Let $\mathcal{D} \in \text{wrPLL}_2^\infty$, and let $\mathcal{D} \rightarrow_{\text{cut}} \mathcal{D}'$ be a cut-elimination step reducing an exponential cut. If $\widehat{\phi}$ is a residue of the exponential flow ϕ then $\mathsf{rk}_{\mathcal{D}'}(\widehat{\phi}) \leq \mathsf{rk}_{\mathcal{D}}(\phi)$.*

PROOF SKETCH. The only interesting case is if $\mathcal{D} \rightarrow_{\text{cut}} \mathcal{D}'$ reduces a cut $\mathsf{r} = \text{c!p-vs-?b}$, and ϕ crosses the active formulas of r with an edge b, as in the rightmost cut-elimination step of Figure 12. Then $\phi = \phi' bc\phi''$, where ϕ' and ϕ'' are directed paths. We have

three subcases. 1) If ϕ does not cross any edge in $\vec{x}, \vec{y}, \vec{z}$ then there is exactly a residue $\widehat{\phi} = \phi' de\phi''$, and $\mathsf{rk}_{\mathcal{D}'}(\widehat{\phi}) < \mathsf{rk}_{\mathcal{D}}(\phi)$. 2) If $\phi = \phi' x_i z_i \phi'' bc\phi'''$, then there are two *distinct* residues $\widehat{\phi}_1 = \phi'$ and $\widehat{\phi}_2 = \phi'''$; indeed, any path $\phi^* = \phi' \psi \phi'''$ cannot be a residue, as it would cross the !-node $!A \in \mathcal{G}(\mathcal{D}) \cap \mathcal{G}(\mathcal{D}')$, which is not crossed by ϕ. 3) If $\phi = \phi' x_i y_i \phi'' abc\phi'''$, and so there is exactly one residue $\widehat{\phi} = \phi' u_i v_i \phi''' de\phi'''$; but then $\mathsf{rk}_{\mathcal{D}'}(\widehat{\phi}) = \mathsf{rk}_{\mathcal{D}}(\phi)$. $\square$

We now define special exponential flows, called *balanced*, which have *at most one* residue along cut-elimination steps reducing exponential cuts different from c!p-vs-c!p.

Definition 42 (Balanced exponential flows). Let $\mathcal{D} \in \text{wrPLL}_2^\infty$. An exponential flow ϕ is **balanced** if every !-node crossed is a p-node, and $\mathsf{p}(\phi, \mathfrak{S}) > \mathsf{b}(\phi[!A])$ for any nwb $\mathfrak{S}$ with principal !-formula $!A$ that is crossed by ϕ.

Proposition 43 (Invariance). *Let $\mathcal{D} \in \text{wrPLL}_2^\infty$ be a coderivation, and let $\mathcal{D} \rightarrow_{\text{cut}} \mathcal{D}'$ be a cut-elimination step reducing an exponential cut $\mathsf{r} \neq \text{c!p-vs-c!p}$. Then, any balanced exponential flow has at most one residue, and it is balanced.*

PROOF SKETCH. The only interesting case is when $\mathcal{D} \rightarrow_{\text{cut}} \mathcal{D}'$ reduces a cut $\mathsf{r} = \text{c!p-vs-?b}$, and ϕ crosses the active formulas of r with an edge b, as in the rightmost cut-elimination step of Figure 12. Since ϕ is balanced, every !-node crossed is a p-node, and so the nodes $!A$ are in the main branch of a nwb $\mathfrak{S}$. Moreover, since $\mathsf{p}(\phi, \mathfrak{S}) > \mathsf{b}(\phi[!A])$ the exponential flow ϕ must be of the form $\phi' x_i y_i \phi'' abc\phi'''$, for some directed paths ϕ', ϕ'', ϕ''', and there is exactly one residue $\widehat{\phi} = \phi' u_i v_i \phi'' de\phi'''$. $\square$

5.3 Shallow cut-elimination strategy

Proving cut-elimination for wrPLL_2^∞ (and rPLL_2^∞) requires infinitary rewriting techniques (see, e.g., [2]). However, if we consider those coderivations of wrPLL_2^∞ with !-free conclusions, we can define cut-elimination strategies that always halt after a *finite* number of steps. This restricted form of cut-elimination suffices for our characterisation results, as computation over binary strings can be duly simulated by cut-elimination over these particular coderivations.

We first classify some special cut rules and introduce a notion of *residue* that tracks cut rules along cut-elimination.

Definition 44 (Cut rules). Let $\mathcal{D} \in \text{wrPLL}_2^\infty$, and let r be a cut. We say that r is **shallow** if $\mathsf{nl}_{\mathcal{D}}(\mathsf{r}) = 0$. It is **bordered** if it is of the form c!p-vs-c!p, c!p-vs-?b, or c!p-vs-?w and its active !-formula is principal for a nwb of $\mathcal{D}$.

Given a cut-elimination step $\mathcal{D} \rightarrow_{\text{cut}} \mathcal{D}'$ and an exponential cut of $\mathcal{D}$, **a residue of** r **(along $\mathcal{D} \rightarrow_{\text{cut}} \mathcal{D}'$)** is a cut rule $\widehat{\mathsf{r}}$ of $\mathcal{D}'$ that satisfies the following condition: either $\widehat{\mathsf{r}} := \mathsf{r}$, if $\mathcal{D} \rightarrow_{\text{cut}} \mathcal{D}'$ does not reduce (or erase) r, or $\widehat{\mathsf{r}}$ is the only cut that is obtained by reducing r and has same active formulas of r (see Figure 9).

Remark 45. Notice that the residue of a exponential cut r along a cut-elimination step is *unique*. Moreover, if the exponential cut is shallow, it has no residue exactly when $\mathsf{r} = \text{c!p-vs-?w}$ and the cut-elimination step reduces r (since r is shallow, it is never erased by reducing another cut $\mathsf{r}' = \text{c!p-vs-?w}$). Finally, observe that every residue of a shallow bordered cut is shallow and bordered.

Figure 12: From left, a cut-elimination step $\mathcal{D} \to_{\text{cut}} \mathcal{D}'$ reducing c!p-vs-?w and c!p-vs-?b, and the corresponding exponential graphs (only the relevant nodes and edges are displayed). Double circles (resp., double edges) represent multiple nodes (resp., multiple edges), while squared nodes are nodes shared by $\mathcal{G}(\mathcal{D})$ and $\mathcal{G}(\mathcal{D}')$. Edges are labelled by letters, and vectors $\vec{x} = x_1, \ldots, x_n$ represent a list of labels, one for each edge.

Definition 46 (Shallow rewriting strategy). Let $\mathcal{D} \in \text{wrPLL}_2^\infty$ be a coderivation of a !-free sequent. The **shallow cut-elimination strategy** iterates $\mathbf{d}(\mathcal{D}) + 1$ times the following two phases[6]:

- **Phase 1.** Reduce all shallow cuts that are not bordered.
- **Phase 2.** Reduce hereditarily all the residues of shallow bordered cuts except c!p-vs-c!p.

We call each iteration of the two phases above a **round**.

We set $\mathcal{D}^0 := \mathcal{D}$ and, for all $1 \leq d \leq \mathbf{d}(\mathcal{D}) + 1$, $\mathcal{D}^d$ to be the coderivation obtained after the d-th round. For all $1 \leq d \leq \mathbf{d}(\mathcal{D}) + 1$ we set $\mathcal{D}_e^d$ as the coderivation obtained from $\mathcal{D}^{d-1}$ by applying **Phase 1**. We write $\mathcal{D}^d \to_{\text{cut}}^{*m} \mathcal{D}_e^d + 1$ to denote that $\mathcal{D}_e^d + 1$ has been obtained in a finite number of steps from $\mathcal{D}^d$ by applying **Phase 1**; $\mathcal{D}_e^d \to_{\text{cut}}^{*e} \mathcal{D}^d$ to denote that $\mathcal{D}^d$ has been obtained in a finite number of steps from $\mathcal{D}_e^d$ by applying **Phase 2**; $\mathcal{D}^d \to_{\text{cut}}^{*r}$ $\mathcal{D}^{d+1} := \mathcal{D}^d \to_{\text{cut}}^{*m} \mathcal{D}_e^{d+1} \to_{\text{cut}}^{*e} \mathcal{D}^{d+1}$.

Termination of the shallow cut-elimination strategy is an immediate consequence of the following technical lemma.

Lemma 47. *Let $\mathcal{D} \in \text{wrPLL}_2^\infty$ with !-free conclusion. Then, the shallow cut-elimination strategy applied to $\mathcal{D}$ satisfies the following properties for every $1 \leq d \leq \mathbf{d}(\mathcal{D}) + 1$:*

(1) $\text{base}(\mathcal{D}^{d-1}) \to_{\text{cut}}^* \text{base}(\mathcal{D}_e^d)$.
(2) either $\mathbf{d}(\mathcal{D}^{d-1}) = 0$ *or* $\mathbf{d}(\mathcal{D}^d) = \mathbf{d}(\mathcal{D}^{d-1}) - 1$.
(3) $\lfloor \mathcal{D}_e^d \rfloor_{\text{rk}(\mathcal{D}_e^d)} \to_{\text{cut}}^* \text{base}(\mathcal{D}^d)$.

PROOF. Item 1 follows from the fact that shallow cuts that are not bordered only affect $\text{base}(\mathcal{D}^{d-1})$, so that $\text{base}(\mathcal{D}^{d-1}) \to_{\text{cut}}^*$ $\text{base}(\mathcal{D}_e^d)$ by Proposition 27. Let us prove Item 2 and Item 3, and let $\mathfrak{S}_1, \ldots, \mathfrak{S}_n$ $(n \geq 0)$ be the nwbs of $\mathcal{D}_e^d$ (and of $\mathcal{D}^{d-1}$) with nesting level 0. Since all shallow cuts are bordered, there are exactly n shallow cuts $r_1, \ldots, r_n$ and the active !-formula of each r_i, say $!A_i$, is principal for $\mathfrak{S}_i$. This means that all !-nodes of $\mathcal{G}(\mathcal{D}_e^d)$ are p-nodes So there is a (possibly repeating) list of balanced exponential flows $\phi_1, \ldots, \phi_n$ of $\mathcal{G}(\mathcal{D}_e^d)$ such that the node $!A_i$ is crossed by ϕ_i.

By Proposition 43, if we reduce a cut $r \neq$ c!p-vs-c!p crossed by a balanced exponential flow we obtain exactly one balanced exponential flow $\widehat{\phi}$. By Definition 44, $\widehat{\phi}$ crosses the residue $\widehat{r}$ of r, which is shallow and bordered by Remark 45.

---

[6]To make the strategy deterministic, we can give priority to the rightmost reducible cut with smallest height. This would ensure that the strategy eventually applies a cut-elimination step to every reducible cut.

We can easily show that, as long as there are shallow bordered cuts in balanced exponential flows there are also cuts $r \neq$ c!p-vs-c!p crossed by them, so if **Phase 2** can only terminate whenever there is hereditarily no (shallow bordered) residue of $r_1, \ldots, r_n$, i.e., if the nwbs $\mathfrak{S}_1, \ldots, \mathfrak{S}_n$ are eventually erased by a c!p-vs-?w by Remark 45. Hence, if $n \geq 0$ then **Phase 2** decreases $\mathcal{D}_e^d$ by 1, and so $\mathbf{d}(\mathcal{D}^d) = \mathbf{d}(\mathcal{D}_e^d) - 1 = \mathbf{d}(\mathcal{D}^{d-1}) - 1$. This proves Item 2.

Finally, let $m \geq 0$ and $\mathcal{D}_e^d = \mathcal{D}_0 \to_{\text{cut}} \mathcal{D}_1 \to_{\text{cut}} \ldots \to_{\text{cut}} \mathcal{D}_m$ be the first m cut-elimination steps of **Phase 2** on $\mathcal{D}_e^d$. Since **Phase 2** only reduces shallow bordered cuts crossed by $\phi_1, \ldots, \phi_n$ and their (unique) residues, by Proposition 41 there are at most $\text{rk}_\mathcal{D}(\phi_i)$ $(\leq \text{rk}(\mathcal{D}_e^d))$ c!p-vs-?b steps involving a c!p rule in (the main branch of) $\mathfrak{S}_i$. Moreover, since c!p-vs-c!p is never reduced, only the first $\text{rk}(\mathcal{D}_e^d)$ c!p rules of $\mathfrak{S}_i$ (from bottom) are affected by **Phase 2**. This means that $\lfloor \mathcal{D}_e^d \rfloor_{\text{rk}(\mathcal{D}_e^d)} = \lfloor \mathcal{D}_0 \rfloor_{\text{rk}(\mathcal{D}_e^d)} \to_{\text{cut}} \mathcal{D}_1' \ldots \to_{\text{cut}} \mathcal{D}_m'$ for some finite approximations $(\mathcal{D}_i')_{1 \leq i \leq m}$ such that $\text{base}(\mathcal{D}_i') = \text{base}(\mathcal{D}_i)$. Moreover, since $\lfloor \mathcal{D}_e^d \rfloor_{\text{rk}(\mathcal{D}_e^d)}$ is a finite approximation, m is bounded by Proposition 27, and so $\lfloor \mathcal{D}_e^d \rfloor_{\text{rk}(\mathcal{D}_e^d)} \to_{\text{cut}}^* \mathcal{D}^*$, for some finite approximation $\mathcal{D}^*$ such thay $\text{base}(\mathcal{D}^*) = \text{base}(\mathcal{D}^d)$. By Item 2 it must be that $\mathbf{d}(\mathcal{D}^*) = 0$, and so $\mathcal{D}^* = \text{base}(\mathcal{D}^*)$. □

THEOREM 48 (TERMINATION). *Let $\mathcal{D} \in \text{wrPLL}_2^\infty$ with !-free conclusion. Then, the shallow cut-elimination strategy applied to $\mathcal{D}$ terminates in a* finite *number of steps returning a cut-free derivation.*

5.4 Polynomial modulus of continuity

In this subsection we establish our characterisation theorems. They rely on a key result, i.e. Lemma 50, which shows that shallow cut-elimination requires a number of steps that can be polynomially bounded w.r.t. the *cosize* of the starting coderivation.

Lemma 49. *Let $\mathcal{D} \in \text{wrPLL}_2^\infty$ be a coderivation of a* !*-free sequent. Then,* $||\mathcal{D}^d||_0 \in O\left(\prod_{i=0}^d \left(||\mathcal{D}^0||_i\right)^{6^{d+1-i}}\right)$ *for all* $0 \leq d \leq \mathbf{d}(\mathcal{D}) + 1$.

PROOF. First, we notice that, since $\mathcal{D}$ is weakly regular and progressing, there is a bound $s^* \geq 0$ on the maximum number of ?-formulas in the conclusion of a c!p rule of $\mathcal{D}$, i.e., $\mathcal{S}(\mathcal{D})$ (see Proposition 27). Moreover, by Proposition 23, we can assume that $s^* \geq 0$ bounds $\mathcal{S}(\mathcal{D}_e^d)$ and $\mathcal{S}(\mathcal{D}^d)$. Hence $\mathcal{S}(\mathcal{D}_e^d)$ and $\mathcal{S}(\mathcal{D}^d)$ will be considered as *constants* throughout this proof.

We prove the statement by induction on $0 \leq d \leq \mathbf{d}(\mathcal{D}) + 1$. The case $d = 0$ is trivial. If $d > 0$ then, by Theorem 48.3, we

have $\lfloor \mathcal{D}_e^d \rfloor_{\mathrm{rk}(\mathcal{D}_e^d)} \rightarrow_{\mathrm{cut}}^* \mathrm{base}(\mathcal{D}^{d+1})$. Then $||\mathcal{D}^d||_0 = \mathrm{base}(\mathcal{D}^d) \in O(\mathcal{S}(\lfloor \mathcal{D}_e^d \rfloor_{\mathrm{rk}(\mathcal{D}_e^d)}) \cdot |\lfloor \mathcal{D}_e^d \rfloor_{\mathrm{rk}(\mathcal{D}_e^d)}|) = O(|\lfloor \mathcal{D}_e^d \rfloor_{\mathrm{rk}(\mathcal{D}_e^d)}|)$ by Proposition 27.2. Moreover, if $\mathfrak{S}_1, \ldots, \mathfrak{S}_n$ are the nwbs of $\mathcal{D}_e^d$ with nesting level 0, since $n, \mathrm{rk}(\mathcal{D}_e^d) \le ||\mathcal{D}_e^d||_0$ and $\mathrm{base}(\mathfrak{S}_i(j)) \le ||\mathcal{D}_e^d||_1$, then
$$|\lfloor \mathcal{D}_e^d \rfloor_{\mathrm{rk}(\mathcal{D}_e^d)}| = |\mathrm{base}(\mathcal{D}_e^d)| + \sum_{i=0}^{n} \sum_{j=0}^{\mathrm{rk}(\mathcal{D}_e^{d+1})} |\mathrm{base}(\mathfrak{S}_i(j))| \in$$
$$O\left(||\mathcal{D}_e^d||_0 + \left(||\mathcal{D}_e^d||_0\right)^2 \cdot ||\mathcal{D}_e^d||_1\right) = O\left(\left(||\mathcal{D}_e^d||_0\right)^2 \cdot ||\mathcal{D}_e^d||_1\right).$$

On the other hand, by Proposition 27.1 we have
$$||\mathcal{D}_e^d||_0 = \mathrm{base}(\mathcal{D}_e^d) \in O(\mathcal{S}(\mathrm{base}(\mathcal{D}^{d-1})^3 \cdot \mathrm{base}(\mathcal{D}^{d-1})^3) =$$
$$= O(\mathcal{S}(\mathrm{base}(\mathcal{D}^{d-1})^3 \cdot ||\mathcal{D}^{d-1}||_0^3) = O\left(\left(||\mathcal{D}^{d-1}||\right)_0^3\right)$$

Finally, we notice that $||\mathcal{D}_e^d||_1 = ||\mathcal{D}^{d-1}||_1 = ||\mathcal{D}^0||_d$ as by Theorem 48.3 the rules of $\mathcal{D}^0$ with nesting level d are unaffected in the first $d-1$ rounds of cut-elimination, and by Theorem 48.2 each round decreases the depth. Then, by inductive hypothesis,
$$||\mathcal{D}^d||_0 \in O\left(\left(||\mathcal{D}_e^d||_0\right)^2 \cdot ||\mathcal{D}_e^d||_1\right) = O\left(\left(||\mathcal{D}_e^d||_0\right)^2 \cdot ||\mathcal{D}^0||_d\right) =$$
$$= O\left(\left(||\mathcal{D}^{d-1}||_0\right)^6 \cdot ||\mathcal{D}^0||_d\right) = O\left(\left(\prod_{i=0}^{d-1} \left(||\mathcal{D}^0||_i\right)^{6^{d-i}}\right)^6 \cdot ||\mathcal{D}^0||_d\right) =$$
$$= O\left(\prod_{i=0}^{d-1} \left(||\mathcal{D}^0||_i\right)^{6^{d+1-i}} \cdot ||\mathcal{D}^0||_d\right) = O\left(\prod_{i=0}^{d} \left(||\mathcal{D}^0||_i\right)^{6^{d+1-i}}\right).$$
$\square$

Lemma 50 (Polynomial modulus of continuity). *Let $\mathcal{D} \in \mathrm{wrPLL}_2^\infty$ be a coderivation of a !-free sequent. Then, for some polynomial $p : \mathbb{N} \to \mathbb{N}$ depending solely on $\mathbf{d}(\mathcal{D})$, $\lfloor \mathcal{D} \rfloor_{p(||\mathcal{D}||)}$ rewrites by the shallow cut-elimination strategy to a cut-free hyp-free derivation.*

PROOF SKETCH. It is a straightforward consequence of Theorem 48.1-3, Lemma 49 and the fact that $\mathrm{rk}(\mathcal{D}_e^d) \in O(||\mathcal{D}^d||_0)$. $\square$

THEOREM 51. *[Soundness] Let $f : (\{0,1\}^*)^n \to \{0,1\}^*$:*
(1) If f is representable in wrPLL_2^∞ then $f \in \mathbf{FP}/\mathrm{poly}$;
(2) If f is representable in rPLL_2^∞ then $f \in \mathbf{FP}$.

PROOF SKETCH. We construct a family of circuits $C = (C_n)_{n \ge 0}$ such that, on input $s = b_1, \ldots, b_n \in \{0,1\}^*$, $C_n(s)$ evaluates the coderivation $\dfrac{\overbrace{\mathsf{S}[] \multimap \mathsf{S}}^{f} \quad \overbrace{\mathsf{S}[]}^{s}}{\mathsf{S}} {\scriptstyle \multimap_e}$ to $\underline{f(s)}$, and returns $f(s)$. The size of C_n is polynomial in n by Lemma 50 and the property $||\lfloor \mathcal{D} \rfloor||_m| \in O(m^{\mathbf{d}(\mathcal{D})+1} \cdot ||\mathcal{D}||^{\mathbf{d}(\mathcal{D})+1})$, which relates the size of the m-truncation of $\mathcal{D}$ and the cosize of $\mathcal{D}$. Therefore, $f \in \mathbf{FP}/\mathrm{poly}$. If moreover f is representable in rPLL_2^∞, then $\underline{f}$ is a regular coderivation, and so the function $n \mapsto C_n$ can be constructed uniformly by a polytime Turing machine. Therefore, $f \in \mathbf{FP}$. $\square$

6 COMPLETENESS

In this section we establish the completeness theorem for wrPLL_2^∞ and rPLL_2^∞ (Theorem 58). To this end we introduce nuPTA_2, a type system designed to express computation with access to bits of streams, and we show that the system can encode polynomial time Turing machines with advice. By a similar reasoning, polynomial time computable functions can be represented in PTA_2, a stream-free subsystem of nuPTA_2. We then translate the type systems into

nuPLL_2 and PLL_2, respectively (Theorem 57), and conclude by the simulation theorem relating the inductive and non-wellfounded proof systems (Theorem 24).

6.1 The type systems PTA_2 and nuPTA_2

The type system nuPTA_2 is a type-theoretical counterpart of nuPLL_2, where the linearity restriction in the second-order rules of Figure 2 is duly reflected by a weaker polymorphism, and modal formulas "$\omega\sigma$" express types of streams. We also introduce PTA_2, the stream-free subsystem of nuPTA_2 corresponding to PLL_2.

Definition 52. [$\Lambda_{\mathrm{stream}}$] We define $\Lambda_{\mathrm{stream}}$ as the set of terms generated by the following grammar:
$$M := x \mid \mathsf{I} \mid \mathrm{let} \; \mathsf{I} = x \; \mathrm{in} \; M \mid M \otimes M \mid \mathrm{let} \; x_1 \otimes x_2 = M \; \mathrm{in} \; M$$
$$\lambda x.M \mid MM \mid \mathbf{M} \mid \mathrm{disc} \mid \mathrm{pop}$$

where x ranges over a countable set of term variables and $\mathbf{M} = \mathbf{M}(0) :: \mathbf{M}(1) :: \ldots$ is a stream of terms. Meta-level substitution for terms, written $M[N/x]$, is defined in the standard way. The reduction rules for $\Lambda_{\mathrm{stream}}$ are the following:

$(\lambda x.M)N \rightarrow_\beta M[N/x]$
$\mathrm{let} \; x_1 \otimes x_2 = M \otimes N \; \mathrm{in} \; P \rightarrow_\beta P[M/x_1, N/x_2]$ $\mathrm{let} \; \mathsf{I} = \mathsf{I} \; \mathrm{in} \; M \rightarrow_\beta M$
$\mathrm{pop} \, \mathbf{M} \rightarrow_\beta hd(\mathbf{M}) \otimes tl(\mathbf{M})$ $\mathrm{disc} \, \mathbf{M} \rightarrow_\beta \mathsf{I}$

and apply to any context, where $hd(\mathbf{M})$ and $tl(\mathbf{M})$ are meta operations returning, respectively, head and tail of $\mathbf{M}$. With $\rightarrow_\beta^*$ we denote the reflexive and transitive closure of $\rightarrow_\beta$.

Type assignment systems based on linear logic typically do not satisfy subject reduction, i.e., preservation of typing under normalisation [27]. A key tool to recast this property is to introduce the so-called *essential types* from [28], whose main feature is to prevent these system expressing forms of sharing and duplication of data, which bring about the failure of subject reduction. Consequently, we will define PTA_2 and nuPTA_2 via essential types.

Definition 53 (PTA_2 and nuPTA_2). The *essential types* are generated by the following grammar:
$$A := X \mid \mathbf{1} \mid \sigma \multimap A \mid \forall X.A \qquad \sigma := A \mid \sigma \otimes \sigma \mid !\sigma \mid \omega\sigma \qquad (2)$$
where X ranges over a countable set of type variables. We denote by $\sigma[\tau/X]$ the meta-level substitution of τ for the free occurrences of the type variable X in σ. A *context* is a set of the form $x_1 : \sigma_1, \ldots, x_n : \sigma_n$ for some $n \ge 0$, where the x_i's are pairwise distinct term variables and σ_i are types. Contexts range over $\Gamma, \Delta, \Sigma, \ldots$. We denote by $!\Gamma$ a context of the form $x_1 : !\sigma_1, \ldots, x_n : !\sigma_n$. The type assignment system for $\Lambda_{\mathrm{stream}}$, called nuPTA_2, derives *judgements* of the form $\Gamma \vdash M : \sigma$ according to the typing rules in Figure 13. The restriction of nuPTA_2 without the typing rules stream, disc and pop is called PTA_2. We write $\Gamma \vdash_{\mathrm{nuPTA}_2} M : \sigma$ (resp. $\Gamma \vdash_{\mathrm{PTA}_2} M : \sigma$) when the judgement $\Gamma \vdash M : \sigma$ is derivable in nuPTA_2 (resp. PTA_2), omitting the subscript when it is clear from the context. If $\mathcal{D}$ is a typing derivation of $\Gamma \vdash M : \sigma$ then we write $\mathcal{D} : \Gamma \vdash M : \sigma$.

Essential types ensure subject reduction for PTA_2 and nuPTA_2.

Proposition 54 (Subject reduction). *Let $\mathcal{D} : \Gamma \vdash M : \sigma$. If $M \rightarrow_\beta N$ then there is $\mathcal{D}'$ such that $\mathcal{D}' : \Gamma \vdash N : \sigma$.*

$$\mathsf{ax}\,\frac{}{x:A\vdash x:A}\qquad \multimap_i\frac{\Gamma,x:\sigma\vdash M:B}{\Gamma\vdash\lambda x.M:\sigma\multimap B}\qquad \multimap_e\frac{\Gamma\vdash M:\sigma\multimap B\quad\Delta\vdash N:\sigma}{\Gamma,\Delta\vdash MN:B}\qquad \otimes_i\frac{\Gamma\vdash M:\sigma\quad\Delta\vdash N:\tau}{\Gamma,\Delta\vdash M\otimes N:\sigma\otimes\tau}\qquad \otimes_e\frac{\Gamma\vdash M:\sigma\otimes\tau\quad\Delta,x:\sigma,y:\tau\vdash P:C}{\Gamma,\Delta\vdash\mathsf{let}\,x\otimes y=M\,\mathsf{in}\,P:C}$$

$$\forall_i\frac{\Gamma\vdash M:A}{\Gamma\vdash M:\forall X.A}\qquad \forall_e\frac{\Gamma\vdash M:\forall X.A}{\Gamma\vdash M:A[B/X]}(\star)\qquad \mathsf{I}_i\frac{}{\vdash\mathsf{I}:\mathbf{1}}\qquad \mathsf{I}_e\frac{\Gamma\vdash N:\mathbf{1}\quad\Delta\vdash M:C}{\Gamma,\Delta\vdash\mathsf{let}\,\mathsf{I}=N\,\mathsf{in}\,M:C}\qquad \mathsf{f!p}\frac{\Gamma\vdash M:\sigma}{!\Gamma\vdash M:!\sigma}\qquad \mathsf{?w}\frac{\Gamma\vdash M:\tau}{\Gamma,x:!\sigma\vdash M:\tau}\qquad \mathsf{?b}\frac{\Gamma,y:\sigma,z:!\sigma\vdash M:\tau}{\Gamma,x:!\sigma\vdash M[x/y,x/z]:\tau}$$

$$\mathsf{stream}\,\frac{\vdash\mathbf{M}:(0):\sigma\quad\vdash\mathbf{M}:(1):\sigma\quad\ldots\quad\vdash\mathbf{M}:(n):\sigma\quad\ldots}{\vdash\mathbf{M}:\omega\sigma}\,\{\mathbf{M}(i)\mid i\in\mathbb{N}\}\text{ is finite}\qquad \mathsf{disc}\,\frac{}{\vdash\mathsf{disc}:\omega\sigma\multimap\mathbf{1}}\qquad \mathsf{pop}\,\frac{}{\vdash\mathsf{pop}:\omega\sigma\multimap\sigma\otimes\omega\sigma}$$

Figure 13: Typing rules for system nuPTA_2 **with** $(\star):=B$ **is** $(!,\omega)$**-free.**

6.2 The completeness theorem

In this subsection we show the completeness of nuPTA_2 and PTA_2 for, respectively, **FP/poly** and **FP**. The proof adapts to our setting the encoding of polynomial time Turing machines from [28, 37].

Definition 55. Booleans, Boolean strings and natural numbers are encoded as follows, for any $n\geq 0$ and $s=b_1\cdots b_n\in\{0,1\}^*$:

$$\underline{1}:=\lambda x.\lambda y.x\otimes y\qquad \underline{n}:=\lambda f.\lambda z.f^n z$$
$$\underline{0}:=\lambda x.\lambda y.y\otimes x\qquad \underline{s}:=\lambda f.\lambda z.f\,\underline{b_n}(f\,\underline{b_{n-1}}(\ldots(f\,\underline{b_1}\,z)\ldots))$$

Booleans can be typed by $\mathbf{B}:=\forall X.(X\otimes X)\multimap(X\otimes X)$. Moreover, for any type A, natural numbers and Boolean strings can be typed, respectively, by $\mathbf{N}[A]:=!(A\multimap A)\multimap A\multimap A$ and $\mathbf{S}[A]:=!(\mathbf{B}\multimap A\multimap A)\multimap A\multimap A$. With $\mathbf{N}[]$ we denote $\mathbf{N}[A]$ for some A, and similarly for $\mathbf{S}[]$. We say that a function $f:(\{0,1\}^*)^n\to\{0,1\}^*$ is *representable* in nuPTA_2 (resp. PTA_2) if there is a term $\underline{f}\in\Lambda_{\mathsf{stream}}$ such that $\vdash\underline{f}:\mathbf{S}[]\multimap\ldots\multimap\mathbf{S}[]\multimap\mathbf{S}$ in nuPTA_2 (resp. PTA_2) and $\underline{f}\,\underline{s_1}\cdots\underline{s_n}\to^*_\beta\underline{f(s_1,\ldots,s_n)}$.

Theorem 56. *Let $f:(\{0,1\}^*)^n\to\{0,1\}^*$:*
(1) *If $f\in\mathbf{FP/poly}$ then f is representable in nuPTA_2;*
(2) *If $f\in\mathbf{FP}$ then f is representable in PTA_2.*

Proof idea. Item 1 boils down to proving that any polynomial Turing machine $\mathcal{M}$ with access to (polynomially many) bits of a Boolean stream can be encoded in nuPTA_2, essentially leveraging on Proposition 6. To this end, we first show that any polynomial $p:\mathbb{N}\to\mathbb{N}$ can be encoded by a term $\vdash\underline{p}:\mathbf{N}[]\to\mathbf{N}[]$. Then, we construct the following typable terms:

- $\vdash\mathsf{length}:\mathbf{S}[]\multimap\mathbf{N}[]$, which returns the length of a string
- $\vdash\mathsf{init}:\mathbf{N}[]\multimap\mathbf{Stream}\multimap\mathbf{TM}$, taking a natural number n and a stream α, and returning a blank tape of length n and an extra tape storing the bits of α
- $\vdash\mathsf{In}:\mathbf{S}[\mathbf{TM}]\multimap\mathbf{TM}\multimap\mathbf{TM}$, taking the input string s and a blank tape, and returning the tape filled with the bits of s
- $\vdash\mathsf{Tr}:\mathbf{TM}\multimap\mathbf{TM}$, encoding the transition function of $\mathcal{M}$
- $\vdash\mathsf{Ext}:\mathbf{TM}\multimap\mathbf{S}$, extracting the output string out of a tape

By composing the above encodings we obtain the following term:

$$s:\mathbf{S}[],n:\mathbf{N}[],m:\mathbf{N}[]\vdash\mathsf{Ext}((n\,\mathsf{Tr})(\mathsf{In}\,s\,(\mathsf{init}\,m\,\alpha))):\mathbf{S}\qquad(3)$$

Intuitively, it iterates n times the transition function on an initial configuration containing a tape of length m storing the input s and an extra tape storing the stream α; after the n-th iteration it returns the output string written on the tape.

Now, let $p(x)$ be a polynomial bounding time and space of $\mathcal{M}$. The term $s:!^k\mathbf{S}[]\vdash\underline{p}[\mathsf{length}\,s/x]:\mathbf{N}[]$ takes the input string s and returns a polynomial in the length of s. By composing this term

along the variables n and m of the term in Equation (3), we obtain a term $s:\mathbf{S}[]\vdash\underline{f}:\mathbf{S}$ representing f.

Item 2 follows directly from Item 1 by stripping away streams from the above encoding. □

We can compare the computational strength of type systems and inductive proof systems based on parsimonious linear logic.

Theorem 57. *Let $f:(\{0,1\}^*)^n\to\{0,1\}^*$:*
(1) *If f is representable in nuPTA_2 then so it is in nuPLL_2;*
(2) *If f is representable in PTA_2 then it is in PLL_2.*

Proof. We define a translation $(\_)^\dagger:\mathsf{nuPTA}_2\to\mathsf{nuPLL}_2$ mapping typing derivations of nuPTA_2 to derivations of nuPLL_2 such that, when restricted to typing derivations of PTA_2, it returns derivations of PLL_2. The translation essentially turns the modality ω to $!$, and the typing rules stream, pop and disc to gadgets of nuPLL_2 containing the inference rules ib!p, ?b and ?w, respectively. Simulation is established by a stronger version of subject reduction: if $\mathcal{D}_1:\Gamma\vdash M_1:\sigma$ and $M_1\to_\beta M_2$ then there is a typing derivation $\mathcal{D}_2:\Gamma\vdash M_2:\sigma$ such that $\mathcal{D}_1^\dagger\to^*_{\mathsf{cut}}\mathcal{D}_2^\dagger$. □

Theorem 58 (Completeness). *Let $f:(\{0,1\}^*)^n\to\{0,1\}^*$:*
(1) *If f is representable in wrPLL_2^∞ then so it is in nuPLL_2;*
(2) *If f is representable in rPLL_2^∞ then it is in PLL_2.*

Proof. For $i\in\{1,2\}$, Item (i) follows from Theorem 56.(i), Theorem 57.(i) and Theorem 24.(i). □

7 CONCLUSION AND FUTURE WORK

This paper builds on a series of recent works aimed at developing implicit computational complexity in the setting of circular and non-wellfounded proof theory [12, 14]. We proved that the non-wellfounded proof systems wrPLL_2^∞ and rPLL_2^∞ capture the complexity classes **FP/poly** and **FP** respectively. We then establish a series of characterisations for various finitary proof systems.

We envisage extending the contributions of this paper, among others, to the following research directions.

Polynomial time over the reals. [31] introduces a characterisation of Ko's class of polynomial time computable functions over real numbers [32] based on parsimonious logic. By employing the rule !b to represent the *pop* operation on streams, this complexity class could be modelled within PLL_2^∞ via cut-elimination as in [2].

Probabilistic complexity. De-randomisation methods showing the inclusion of the complexity class **BPP** (bounded-error probabilistic polynomial time) in **FP/poly** suggest that this class can be characterised within wrPLL_2^∞. Challenges are expected, since **BPP**

is defined by explicit (error) bounds, as observed in [36] (so, not entirely in the style of ICC), but we conjecture that error bounds can be traded for appropriate global proof-theoretic conditions on wrPLL$_2^\infty$ that restrict computationally the access to streams.

Logarithmic Space. In [39, 40] the authors characterize the complexity classes **L** (logarithmic space problems) and its non-uniform counterpart **L**/poly (problems decided by polynomial size branching programs) by stripping away second-order quantifiers from their proof systems capturing **P** and **P**/poly. We expect that a similar result can be obtained for our non-wellfounded proof systems.

Non-uniform Proofs-as-Processes. Processes such as a scheduler sorting tasks among a (finite) set of servers according to a predetermined order (e.g., a token ring of servers) may easily be modelled by nwbs, making wrPLL$_2^\infty$ appealing for the study of the proofs-as-processes correspondence and its applications [1, 11, 17, 42, 47].

ACKNOWLEDGMENTS

REFERENCES

[1] Samson Abramsky. 1994. Proofs as processes. *Theoretical Computer Science* 135, 1 (1994), 5–9. https://doi.org/10.1016/0304-3975(94)00103-0

[2] Matteo Acclavio, Gianluca Curzi, and Giulio Guerrieri. 2024. Infinitary cut-elimination via finite approximations. In *Computer Sience Logic (CSL 2024), proceedings (LIPIcs, Vol. 288)*. https://doi.org/10.4230/LIPIcs.CSL.2024.8 arXiv:2308.07789 To appear.

[3] Sanjeev Arora and Boaz Barak. 2009. *Computational Complexity: A Modern Approach.* Cambridge University Press. https://doi.org/10.1017/CBO9780511804090

[4] David Baelde, Amina Doumane, and Alexis Saurin. 2016. Infinitary Proof Theory: the Multiplicative Additive Case. In *25th EACSL Annual Conference on Computer Science Logic, CSL 2016, August 29 - September 1, 2016, Marseille, France (LIPIcs, Vol. 62)*, Jean-Marc Talbot and Laurent Regnier (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 42:1–42:17. https://doi.org/10.4230/LIPIcs.CSL.2016.42

[5] Patrick Baillot. 2015. On the expressivity of elementary linear logic: Characterizing Ptime and an exponential time hierarchy. *Inf. Comput.* 241 (2015), 3–31. https://doi.org/10.1016/j.ic.2014.10.005

[6] Hendrik Pieter Barendregt. 1981. *The Lambda Calculus: Its Syntax and Semantics.* New York, N.Y.: Sole distributors for the U.S.A. and Canada, Elsevier Science Pub. Co.

[7] Stephen Bellantoni and Stephen Cook. 1992. A New Recursion-Theoretic Characterization of the Polytime 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

[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] Stefano Berardi and Makoto Tatsuta. 2019. Classical System of Martin-Lof's Inductive Definitions is not Equivalent to Cyclic Proofs. *Log. Methods Comput. Sci.* 15, 3 (2019). https://doi.org/10.23638/LMCS-15(3:10)2019

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

[11] Luís Caires and Frank Pfenning. 2010. Session types as intuitionistic linear propositions. In *International Conference on Concurrency Theory*. Springer, 222–236.

[12] Gianluca Curzi and Anupam Das. 2022. Cyclic Implicit Complexity. In *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science* (Haifa, Israel) *(LICS '22)*. Association for Computing Machinery, New York, NY, USA, Article 19, 13 pages. https://doi.org/10.1145/3531130.3533340

[13] Gianluca Curzi and Anupam Das. 2022. Cyclic Implicit Complexity. In *LICS '22: 37th Annual ACM/IEEE Symposium on Logic in Computer Science, Haifa, Israel, August 2 - 5, 2022*, Christel Baier and Dana Fisman (Eds.). ACM, 19:1–19:13. https://doi.org/10.1145/3531130.3533340

[14] Gianluca Curzi and Anupam Das. 2023. Non-Uniform Complexity via Non-Wellfounded Proofs. In *31st EACSL Annual Conference on Computer Science Logic,*

[15] *CSL 2023, February 13-16, 2023, Warsaw, Poland (LIPIcs, Vol. 252)*, Bartek Klin and Elaine Pimentel (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 16:1–16:18. https://doi.org/10.4230/LIPIcs.CSL.2023.16

[15] Gianluca Curzi and Luca Roversi. 2020. A type-assignment of linear erasure and duplication. *Theor. Comput. Sci.* 837 (2020), 26–53. https://doi.org/10.1016/J.TCS.2020.05.001

[16] Vincent Danos and Jean-Baptiste Joinet. 2003. and elementary time. *Inf. Comput.* 183, 1 (2003), 123–137. https://doi.org/10.1016/S0890-5401(03)00010-5

[17] Ornela Dardha, Elena Giachino, and Davide Sangiorgi. 2017. Session types revisited. *Information and Computation* 256 (2017), 253–286. https://doi.org/10.1016/j.ic.2017.06.002

[18] Anupam Das. 2020. A circular version of Gödel's T and its abstraction complexity. *CoRR* abs/2012.14421 (2020). arXiv:2012.14421 https://arxiv.org/abs/2012.14421

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

[20] Anupam Das. 2021. On the Logical Strength of Confluence and Normalisation for Cyclic Proofs. In *6th International Conference on Formal Structures for Computation and Deduction, FSCD 2021, July 17-24, 2021, Buenos Aires, Argentina (Virtual Conference) (LIPIcs, Vol. 195)*, Naoki Kobayashi (Ed.). Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 29:1–29:23. https://doi.org/10.4230/LIPIcs.FSCD.2021.29

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

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

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

[24] Abhishek De and Alexis Saurin. 2019. Infinets: The Parallel Syntax for Non-wellfounded Proof-Theory. In *Automated Reasoning with Analytic Tableaux and Related Methods - 28th International Conference, TABLEAUX 2019, London, UK, September 3-5, 2019, Proceedings (Lecture Notes in Computer Science, Vol. 11714)*, Serenella Cerrito and Andrei Popescu (Eds.). Springer, 297–316. https://doi.org/10.1007/978-3-030-29026-9_17

[25] Thomas Ehrhard, Farzad Jafarrahmani, and Alexis Saurin. 2021. On relation between totality semantic and syntactic validity. In *5th International Workshop on Trends in Linear Logic and Applications (TLLA 2021)*. Rome (virtual), Italy. https://hal-lirmm.ccsd.cnrs.fr/lirmm-03271408

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

[27] Marco Gaboardi and Simona Ronchi Della Rocca. 2007. A Soft Type Assignment System for *lambda* -Calculus. In *Computer Science Logic, 21st International Workshop, CSL 2007, 16th Annual Conference of the EACSL, Lausanne, Switzerland, September 11-15, 2007, Proceedings (Lecture Notes in Computer Science, Vol. 4646)*, Jacques Duparc and Thomas A. Henzinger (Eds.). Springer, 253–267. https://doi.org/10.1007/978-3-540-74915-8_21

[28] Marco Gaboardi and Simona Ronchi Della Rocca. 2009. From light logics to type assignments: A case study. *Logic Journal of the IGPL* 17 (09 2009). https://doi.org/10.1093/jigpal/jzp019

[29] Jean-Yves Girard. 1994. Light Linear Logic. In *Logical and Computational Complexity. Selected Papers. Logic and Computational Complexity, International Workshop LCC '94, Indianapolis, USA, 13-16 October 1994 (Lecture Notes in Computer Science, Vol. 960)*, Daniel Leivant (Ed.). Springer, 145–176. https://doi.org/10.1007/3-540-60178-3_83

[30] Jean-Yves Girard. 1998. Light Linear Logic. *Information and Computation* 143, 2 (1998), 175–204. https://doi.org/10.1006/inco.1998.2700

[31] Emmanuel Hainry, Damiano Mazza, and Romain Péchoux. 2020. Polynomial Time over the Reals with Parsimony. In *Functional and Logic Programming - 15th International Symposium, FLOPS 2020, Akita, Japan, September 14-16, 2020, Proceedings (Lecture Notes in Computer Science, Vol. 12073)*, Keisuke Nakano and Konstantinos Sagonas (Eds.). Springer, 50–65. https://doi.org/10.1007/978-3-030-59025-3_4

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

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

[34] Yves Lafont. 2004. Soft linear logic and polynomial time. *Theor. Comput. Sci.* 318, 1-2 (2004), 163–180. https://doi.org/10.1016/j.tcs.2003.10.018

[35] Yves Lafont. 2004. Soft linear logic and polynomial time. *Theoretical Computer Science* 318, 1 (2004), 163–180. https://doi.org/10.1016/j.tcs.2003.10.018 Implicit Computational Complexity.

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

[37] Harry G. Mairson and Kazushige Terui. 2003. On the Computational Complexity of Cut-Elimination in Linear Logic. In *Italian Conference on Theoretical Computer Science*.

[38] Damiano Mazza. 2014. Non-uniform Polytime Computation in the Infinitary Affine Lambda-Calculus. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 8573)*, Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias (Eds.). Springer, 305–317. https://doi.org/10.1007/978-3-662-43951-7_26

[39] Damiano Mazza. 2015. Simple Parsimonious Types and Logarithmic Space. In *24th EACSL Annual Conference on Computer Science Logic, CSL 2015, September 7-10, 2015, Berlin, Germany (LIPIcs, Vol. 41)*, Stephan Kreutzer (Ed.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 24–40. https://doi.org/10.4230/LIPIcs.CSL.2015.24

[40] Damiano Mazza and Kazushige Terui. 2015. Parsimonious Types and Non-uniform Computation. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 9135)*, Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann (Eds.). Springer, 350–361. https://doi.org/10.1007/978-3-662-47666-6_28

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

[42] Fabrizio Montesi and Marco Peressotti. 2021. Linear Logic, the π-calculus, and their Metatheory: A Recipe for Proofs as Processes. arXiv:2106.11818 [cs.LO]

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

[44] Luca Roversi and Luca Vercelli. 2010. Safe Recursion on Notation into a Light Logic by Levels. In *Proceedings International Workshop on Developments in Implicit Computational complExity, DICE 2010, Paphos, Cyprus, 27-28th March 2010 (EPTCS, Vol. 23)*, Patrick Baillot (Ed.). 63–77. https://doi.org/10.4204/EPTCS.23.5

[45] Alex Simpson. 2017. Cyclic Arithmetic Is Equivalent to Peano Arithmetic. In *Foundations of Software Science and Computation Structures - 20th International Conference, FOSSACS 2017, Held as Part of European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings (Lecture Notes in Computer Science, Vol. 10203)*, Javier Esparza and Andrzej S. Murawski (Eds.). 283–300. https://doi.org/10.1007/978-3-662-54458-7_17

[46] A. S. Troelstra and H. Schwichtenberg. 2000. *Basic Proof Theory* (2 ed.). Cambridge University Press. https://doi.org/10.1017/CBO9781139168717

[47] Philip Wadler. 2014. Propositions as sessions. *Journal of Functional Programming* 24, 2-3 (2014), 384–418.

A   PROOFS OF SECTION 2

Proposition 6 (See, e.g., [14]). $\mathbf{FP}/\text{poly} = \mathbf{FP}(\mathbb{R})$.

PROOF SKETCH. For the left-right inclusion, let $p(n)$ be a poly-nomial 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 $\mathbf{FP}$.

For the right-left inclusion, notice that a polynomial-time ma-chine 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 computa-tion in $\mathbf{FP}/\text{poly}$.   □

B   PROOFS OF SECTION 4

B.1   Some properties of wrPLL_2^∞ and rPLL_2^∞

The sets rPLL_2^∞ and wrPLL_2^∞ are the non-wellfounded counterparts of PLL_2 and nuPLL_2, respectively. Indeed, we have the following correspondence via the translations $(\cdot)^\circ$ and $(\cdot)^\bullet$ in Figure 8.

Proposition 59. *The following statements hold:*

- *If $\mathcal{D} \in \text{PLL}_2$ (resp. $\mathcal{D} \in \text{nuPLL}_2$) with conclusion Γ, then $\mathcal{D}^\circ \in \text{rPLL}_2^\infty$ (resp. $\mathcal{D}^\bullet \in \text{wrPLL}_2^\infty$) with conclusion Γ, and every c!p in $\mathcal{D}^\circ$ (resp. $\mathcal{D}^\bullet$) belongs to a nwb.*
- *If $\mathcal{D}' \in \text{rPLL}_2^\infty$ (resp. $\mathcal{D}' \in \text{wrPLL}_2^\infty$) and every c!p in $\mathcal{D}'$ belongs to a nwb, then there is $\mathcal{D} \in \text{PLL}_2$ (resp. $\mathcal{D} \in \text{nuPLL}_2$) such that $\mathcal{D}^\circ = \mathcal{D}'$ (resp. $\mathcal{D}^\bullet = \mathcal{D}'$).*

PROOF. The first statement is proven by induction on $\mathcal{D} \in \text{PLL}_2$ (resp. $\mathcal{D} \in \text{nuPLL}_2$). The second is by proven by induction on $\mathbf{d}(\mathcal{D})$, since $\mathbf{d}(\mathcal{D})$ is finite by Proposition 32.   □

Definition 60. A coderivation $\mathcal{D}$ in PLL_2^∞ is **weakly progressing** if every infinite branch contains infinitely many right premises of c!p-rules. We define wpPLL_2^∞ as the set of weakly progressing coderivations of PLL_2^∞.

Progressing and weak progressing coincide in finitely expandable coderivations.

Lemma 61. *Let $\mathcal{D} \in \text{PLL}_2^\infty$ be finitely expandable. If $\mathcal{D} \in \text{wpPLL}_2^\infty$ then any infinite branch contains the main branch of a nwb. Moreover, $\mathcal{D} \in \text{pPLL}_2^\infty$ if and only if $\mathcal{D} \in \text{wpPLL}_2^\infty$.*

PROOF. Let $\mathcal{D} \in \text{wpPLL}_2^\infty$ be finitely expandable, and let $\mathcal{B}$ be an infinite branch in $\mathcal{D}$. By finite expandability there is $h \in \mathbb{N}$ such that $\mathcal{B}$ contains no conclusion of a cut or ?b with height greater than h. Moreover, by weak progressing condition there is an infinite sequence $h \le h_0 < h_1 < \ldots < h_n < \ldots$ such that the sequent of $\mathcal{B}$ at height h_i has shape $?\Gamma_i, !A_i$. By inspecting the rules in Figure 2, each such $?\Gamma_i, !A_i$ can be the conclusion of either a ?w or a c!p (with right premise $?\Gamma_i, !A_i$). So, there is a k large enough such that, for any $i \ge k$, only the latter case applies (and, in particular, $\Gamma_i = \Gamma$ and

$A_i = A$ for some Γ, A). Therefore, h_k is the root of a nwb. This also shows $\mathcal{D} \in \text{pPLL}_2^\infty \subseteq \text{wpPLL}_2^\infty$.   □

Proposition 28. *Let $\mathcal{D} \in \text{pPLL}_2^\infty$ be finitely expandable. There is a prebar $\mathcal{V} \subseteq \{1, 2\}^*$ of $\mathcal{D}$ such that each $v \in \mathcal{V}$ is the root of a nwb.*

PROOF. A straightforward consequence of Lemma 61.   □

Proposition 62. *It is $\mathbf{NL}$-decidable if a regular coderivation is in rPLL_2^∞.*

PROOF. A regular coderivation is represented by a finite cyclic graph. By Lemma 61 checking progressiveness comes down to checking that no branch has infinitely many occurrences of a par-ticular rule, which in turn reduces to checking acyclicity for this graph (see [13]). We conclude since checking acyclicity is a well-known $\mathbf{coNL}$ problem, and $\mathbf{coNL} = \mathbf{NL}$[3].   □

B.2   Proofs of Subsection 4.4

Lemma 63. *Let $\mathcal{D} \in \text{nuPLL}_2$ (resp. $\mathcal{D} \in \text{PLL}_2$) have !-free conclu-sion. Then*

- *(1) either $\mathcal{D}$ is cut-free or it has a cut rule different from f!p-vs-f!p and ib!p-vs-ib!p.*
- *(2) for any cut-elimination sequence σ rewriting $\mathcal{D}$ to a cut-free derivation $\widehat{\mathcal{D}}$ there is another cut-elimination sequence $\widehat{\sigma}$ rewriting $\mathcal{D}$ to $\widehat{\mathcal{D}}$ that never applies the steps f!p-vs-f!p and ib!p-vs-ib!p.*

PROOF. Concerning Item 1, suppose towards contradiction that there is a derivation $\mathcal{D} \in \text{nuPLL}_2$ that is not cut-free and whose cut rules are all of the form f!p-vs-f!p and ib!p-vs-ib!p. We now establish the following facts:

- There is a sequent in $\mathcal{D}$ containing a !-formula
- If the premise of a rule r in $\mathcal{D}$ contains ! than so the conclu-sion of r does.

The first fact follows from the assumption that there is at least one cut rule and that all cuts contain a !-formula. Concerning the second fact we have two cases. If r = cut then it is either f!p-vs-f!p or ib!p-vs-ib!p, and in both cases r contains a !-formula in the conclusion. Otherwise, r is not a cut, and the property follows by inspecting the rules of nuPLL_2, recalling that instantiation in the $\exists$ rule requires !-freeness. So, the conclusion of $\mathcal{D}$ must contain a !, contradicting our assumptions.

Let us now prove Item 2. On the one hand, it is easy to check that the cut-elimination steps f!p-vs-f!p and ib!p-vs-ib!p commute with any other cut-elimination step, and so we can rewrite σ into another sequence $\widehat{\sigma}$ where all such cut-elimination steps are postponed. This means that there is a derivation $\mathcal{D}'$ in $\widehat{\sigma}$ such that $\mathcal{D} \rightarrow_{\text{cut}}^* \mathcal{D}'$ is free of the cut-elimination steps f!p-vs-f!p and ib!p-vs-ib!p, and $\mathcal{D}' \rightarrow_{\text{cut}}^* \widehat{\mathcal{D}}$ only contains those steps. However, since $\widehat{\mathcal{D}}$ is cut-free, by Lemma 63.1 it must be that $\mathcal{D}' = \widehat{\mathcal{D}}$.   □

Lemma 64. *Let $\mathcal{D}_1 \in \text{nuPLL}_2$ (resp. $\mathcal{D}_1 \in \text{PLL}_2$). If $\mathcal{D}_1 \rightarrow_{\text{cut}} \mathcal{D}_2$ by applying a cut-elimination step different from f!p-vs-f!p and ib!p-vs-ib!p then $\mathcal{D}^\bullet \rightarrow_{\text{cut}} \mathcal{D}_2^\bullet$.*

THEOREM 24. *[Simulation] Let $f : (\{0, 1\}^*)^n \rightarrow \{0, 1\}^*$.*

(1) If f is representable in nuPLL_2, then so it is in wrPLL_2^∞.

(2) *If f is representable in* PLL_2, *then so it is in* rPLL_2^∞.

PROOF. We only prove Item 1, as 2 is proven similarly. Let $\mathcal{D}$: $\mathsf{S}[\,] \multimap^{n \geq 0} \multimap \mathsf{S}[\,] \multimap \mathsf{S}$ represent $f : (\{0,1\}^*)^n \to \{0,1\}^*$ in nuPLL_2 and let $x_1, \ldots, x_n \in \{0,1\}^*$. This means that the reduction in Figure 5 holds for $\mathbf{T}_1 = \ldots = \mathbf{T}_n = \mathsf{S}[\,]$ and $\mathbf{T} = \mathsf{S}$. Let $\sigma := \mathcal{D}_0 \to_{\mathsf{cut}} \mathcal{D}_1 \to_{\mathsf{cut}} \ldots \to_{\mathsf{cut}} \mathcal{D}_n = \underline{f(s_1, \ldots, s_n)}$ be such a reduction sequence, and notice that $\underline{f(s_1, \ldots, s_n)}$ is cut-free by definition. By Lemma 63.2 we can assume, w.l.o.g., that σ never applies the cut-elimination steps f!p-vs-f!p and ib!p-vs-ib!p. We conclude by applying Lemma 64, observing that $\underline{s}^\bullet = \underline{s}$ for any binary string $s \in \{0,1\}^*$. □

B.3 Proofs of Subsection 4.5

Proposition 27 (Cubic bound). *Let $\mathcal{D}$ be an open derivation and let $\mathsf{S}(\mathcal{D})$ be the maximum number of ?-formulas in the conclusion of a c!p rule of $\mathcal{D}$. If $\mathcal{D} = \mathcal{D}_0 \to_{\mathsf{cut}} \cdots \to_{\mathsf{cut}} \mathcal{D}_n$ then:*

(1) n and $|\mathcal{D}_i|$ are in $O(\mathsf{S}(\mathcal{D})^3 \cdot |\mathcal{D}|^3)$ for any $i \in \{0, \ldots, n\}$.
(2) If only principal cut-elimination steps are applied then n and $|\mathcal{D}_i|$ are in $O(\mathsf{S}(\mathcal{D}) \cdot |\mathcal{D}|)$ for any $i \in \{0, \ldots, n\}$.
(3) If the reduction sequence is maximal then $\mathcal{D}_n$ is cut free.

PROOF. For $\mathcal{D}$ an open derivation, let $\mathsf{C}(\mathcal{D})$ be the number of c!p in $\mathcal{D}$ and $\mathsf{H}(\mathcal{D})$ be the sum of the sizes of all subderivations of $\mathcal{D}$ whose root is the conclusion of a cut rule. If $\mathcal{D} \to_{\mathsf{cut}} \mathcal{D}'$ via:

- a commutative cut-elimination step (Figure 4), then $\mathsf{C}(\mathcal{D}) = \mathsf{C}(\mathcal{D}')$, $|\mathcal{D}| = |\mathcal{D}'|$ and $\mathsf{H}(\mathcal{D}) > \mathsf{H}(\mathcal{D}')$;
- a multiplicative cut-elimination step, then $\mathsf{C}(\mathcal{D}) = \mathsf{C}(\mathcal{D}')$ and $|\mathcal{D}| > |\mathcal{D}'|$;
- an exponential cut-elimination step (Figure 9), then and $\mathsf{C}(\mathcal{D}) > \mathsf{C}(\mathcal{D}')$

Since the lexicographic order over the tuple $(\mathsf{C}(\mathcal{D}), |\mathcal{D}|, \mathsf{H}(\mathcal{D}))$ is wellfounded, we conclude that there is no infinite sequence $(\mathcal{D}_i)_{i \in \mathbb{N}}$ such that $\mathcal{D}_0 = \mathcal{D}$ and $\mathcal{D}_i \to_{\mathsf{cut}} \mathcal{D}_{i+1}$.

Now, let $\mathcal{D} = \mathcal{D}_0 \to_{\mathsf{cut}} \cdots \to_{\mathsf{cut}} \mathcal{D}_n$ with $\mathcal{D}_n$ normal. First, we show that the number n_p of its principal cut-elimination steps is bounded by $\mathsf{W}(\mathcal{D}) := \mathsf{S}(\mathcal{D}) \cdot \mathsf{C}(\mathcal{D}) + \mathsf{M}(\mathcal{D})$, where $\mathsf{M}(\mathcal{D})$ is the number of inference rules different from c!p in $\mathcal{D}$. This boils down to showing that $\mathcal{D}' \to_{\mathsf{cut}} \mathcal{D}''$ implies $\mathsf{W}(\mathcal{D}'') < \mathsf{W}(\mathcal{D}')$. Indeed:

- every cut-elimination step can only decrease $\mathsf{S}(\mathcal{D})$
- every multiplicative cut-elimination step strictly decreases $\mathsf{M}(\mathcal{D})$ and does not increase $\mathsf{C}(\mathcal{D})$
- the exponential steps c!p-vs-?w and c!p-vs-?b strictly decrease $\mathsf{C}(\mathcal{D})$ and do not increase $\mathsf{C}(\mathcal{D})$
- if $\mathcal{D}' \to_{\mathsf{cut}} \mathcal{D}''$ is obtained by applying a c!p-vs-?b step then

$$\begin{aligned}
\mathsf{W}(\mathcal{D}'') &:= \mathsf{S}(\mathcal{D}'') \cdot \mathsf{C}(\mathcal{D}'') + \mathsf{M}(\mathcal{D}'') \\
&\leq \mathsf{S}(\mathcal{D}') \cdot \mathsf{C}(\mathcal{D}'') + (\mathsf{M}(\mathcal{D}') - 1 + \mathsf{S}(\mathcal{D}')) \\
&= \mathsf{S}(\mathcal{D}') \cdot (\mathsf{C}(\mathcal{D}') - 1) + \mathsf{M}(\mathcal{D}') - 1 + \mathsf{S}(\mathcal{D}') \\
&= \mathsf{S}(\mathcal{D}') \cdot \mathsf{C}(\mathcal{D}') + \mathsf{M}(\mathcal{D}') - 1 < \mathsf{W}(\mathcal{D}')
\end{aligned}$$

At the same time, the number n_c^i of commutative steps performed after the i-th principal is bounded by the square of the maximum size of the proof during rewriting, which can be bounded by $\mathsf{W}(\mathcal{D})$.

Hence, we have:

$$\begin{aligned}
n &= n_p + \sum_{i=1}^{n_p} n_c^i \leq n_p + n_p \max_i\{n_c^i\} \\
&\leq n_p \left(\max_i\{n_c^i\} + 1\right) \leq \mathsf{W}(\mathcal{D}) \cdot (\mathsf{W}(\mathcal{D})^2 + 1) \\
&\leq 2\mathsf{W}(\mathcal{D})^3
\end{aligned}$$

We conclude as $\mathsf{W}(\mathcal{D}) \in O(\mathsf{S}(\mathcal{D}) \cdot |\mathcal{D}|)$ and $|\mathcal{D}_i| \leq \mathsf{W}(\mathcal{D}_i) \leq \mathsf{W}(\mathcal{D})$. □

Lemma 65 ([2]). $\to_{\mathsf{cut}}$ *over open derivations is strongly normalizing and confluent.*

PROOF. Strong normalisation is a consequence of Proposition 27. Moreover, since cut-elimination $\to_{\mathsf{cut}}$ is strongly normalizing over open derivations and it is locally confluent by inspection of critical pairs, by Newman's lemma it is also confluent. □

C PROOF OF SECTION 5

C.1 Proofs of Subsection 5.2

Proposition 66 (Basic properties). *Let $\mathcal{D} \in \mathsf{wrPLL}_2^\infty$ be a coderivation. For every exponential flow ϕ of $\mathcal{G}(\mathcal{D})$:*

(1) ϕ is finite and $\mathsf{b}(\phi) \leq \mathsf{rk}(\mathcal{D}) \in \mathbb{N}$.
(2) if ϕ crosses a nwb $\mathfrak{S}$ then, for every $n \geq 1$, there is an exponential flow ϕ^n such that:
- *ϕ^n crosses all nodes of ϕ*
- *$\mathsf{p}(\phi^n, \mathfrak{S}) = n$*

PROOF. Let us prove Item 1. By Proposition 28 there is a decomposition prebar $\mathsf{border}(\mathcal{D}) = \{v_1, \ldots, v_n\}$ so that each $v_i \in \mathsf{border}(\mathcal{D})$ is the root of a nwb $\mathfrak{S}_i$. By definition, we have $\mathbf{nl}_{\mathcal{D}}(\mathfrak{S}_i) = 0$. Hence, the nodes of $\mathcal{G}(\mathcal{D})$ are:

- the occurrences of exponential formulas in $\mathsf{base}(\mathcal{D})$
- for all $1 \leq i \leq n$, the occurrences of the exponential formulas in the main branch of $\mathfrak{S}_i$.

Since $\mathsf{base}(\mathcal{D})$ is finite, it has finitely many nodes in $\mathcal{G}(\mathcal{D})$. Also, by definition, only finitely many nodes of the main branch of each $\mathfrak{S}_i$ can be crossed by the same exponential flow. Since $\mathcal{G}(\mathcal{D})$ is acyclic, this means that any exponential flow must be finite.

Moreover, we observe that only finitely many formulas with nesting level 0 in $\mathcal{D}$ are principal formulas for a b rule, and so $\mathcal{G}(\mathcal{D})$ has finitely many b-nodes.

Let us now prove Item 2. It suffices to show how to construct ϕ^{n+1} from ϕ^n for all $n \geq 1$. We first notice that ϕ^n can only cross in $\mathfrak{S}$ occurrences of the same ?-formula, say ?A. So, let $r_1, r_2, \ldots$ the infinite sequence of c!p rules in the main branch of $\mathfrak{S}$ (starting from the bottommost one), and let a_i (resp., b_i) be the edge of $\mathcal{G}(\mathcal{D})$ connecting the occurrence of ?A (resp., !B) in the conclusion of r_i (resp., r_{i+1}) to the occurrence of ?A (resp., !B) in the conclusion of r_{i+1} (resp., r_i). Finally, let b_i be the edge of $\mathcal{G}(\mathcal{D})$ connecting the occurrences of ?A and !B in the conclusion of r_i. We have that ϕ^n must be a directed path of the form $\phi' a_1 \ldots a_n b_n c_n \ldots c_1 \phi''$, for some directed paths ϕ', ϕ''. Then, we set $\phi^{n+1} := \phi' a_1 \ldots a_n a_{n+1} b_{n+1} c_{n+1} \ldots c_1 \phi''$. □

Proposition 67. *Let $\mathcal{D} \in \mathsf{wrPLL}_2^\infty$ be a coderivation with !-free conclusion and with only exponential cuts. Then, for every exponential flow ϕ:*

(1) ϕ ends at a w-node.

(2) If ϕ crosses a p-node, and every !-node crossed is a p-node, then ϕ crosses a cut rule $r \neq$ c!p-vs-c!p.

PROOF. Let us prove Item 1. By assumption, an exponential flow cannot end at the conclusion of $\mathcal{D}$ and, since instantiations of $\exists$ are !, ?-free, it cannot end at an active formula of a $\exists$ rule. So, since exponential flows are maximal paths, they must end at the the principal formula of a ?w rule or at an active formula of a multiplicative rule. The latter case is impossible, as $\mathcal{D}$ has a !-free conclusion and only exponential cuts.

Concerning Item 1, Proposition 66.1 ϕ is finite. Let us denote with $\sharp(\phi')$ the length of a path ϕ'. Among the (finitely many) !-nodes of ϕ there exists one, say $!A$, with minimal $\sharp(\phi[!A])$. From the fact that every !-node is a p-node, i.e., every !-formula is in fact the principal formula of a c!p rule in the main branch of a nwb, we infer that $!A$ is both in the conclusion of a nwb and an active formula of a cut r. Moreover, by definition, ϕ also crosses the dual active formula of r, i.e., $?A^\perp$. If the latter were in the conclusion of a c!p rule (and so in the conclusion of a nwb), by definition of exponential flow, ϕ would also cross its principal !-formula, say $!B$. But then $\sharp(\phi[!B]) < \sharp(\phi[!A])$, contradicting our hypothesis. So, it must be that $r \neq$ c!p-vs-c!p. □

Proposition 41. *Let $\mathcal{D} \in \text{wrPLL}_2^\infty$, and let $\mathcal{D} \rightarrow_{\text{cut}} \mathcal{D}'$ be a cut-elimination step reducing an exponential cut. If $\widehat{\phi}$ is a residue of the exponential flow ϕ then $\text{rk}_{\mathcal{D}'}(\widehat{\phi}) \leq \text{rk}_{\mathcal{D}}(\phi)$.*

PROOF. The statement is straightforward if $\mathcal{D} \rightarrow_{\text{cut}} \mathcal{D}'$ is a commuting cut-elimination step or if it reduces a cut that is not crossed by ϕ. Indeed, notice that ϕ cannot be erased by the cut-elimination step, since nodes crossed by ϕ have nesting level 0. So, let us suppose that the cut r reduced by $\mathcal{D} \rightarrow_{\text{cut}} \mathcal{D}'$ is crossed by ϕ. There are four cases:

- If $r = $ c!p-vs-ax then there is exactly one residue $\widehat{\phi}$ and $\text{rk}_{\mathcal{D}'}(\widehat{\phi}) = \text{rk}_{\mathcal{D}}(\phi)$.
- $r = $ c!p-vs-?w and ϕ crosses the active formulas of r with an edge b, as in the leftmost cut-elimination step of Figure 12. We have three subcases. If ϕ does not cross any edge in $\vec{x}, \vec{y}, \vec{z}$, then $\phi = \phi'ab$, where ϕ' is a directed path. But then there is no residue, so the statement holds vacuously. Otherwise, there is exactly one residue $\widehat{\phi}$ and either $\phi = \phi'x_iz_i\phi''b$ or $\phi = \phi'x_iy_i\phi''ab\phi''$, where ϕ', ϕ'', ϕ''' are directed paths. In any case $\widehat{\phi} = \phi'$, so $\text{rk}_{\mathcal{D}'}(\widehat{\phi}) \leq \text{rk}_{\mathcal{D}}(\phi)$.
- $r = $ c!p-vs-?b and ϕ crosses the active formulas of r with an edge b, as in the rightmost cut-elimination step of Figure 12. Let $?\Gamma = ?C_1, \ldots, ?C_n$. Then $\phi = \phi'bc\phi''$, where ϕ', ϕ'' are directed paths. We have three subcases:
  - If ϕ does not cross any edge in $\vec{x}, \vec{y}, \vec{z}$ then there is exactly one residue $\widehat{\phi} = \phi'de\phi''$, and $\text{rk}_{\mathcal{D}'}(\widehat{\phi}) < \text{rk}_{\mathcal{D}}(\phi)$.
  - If $\phi = \phi'x_iz_i\phi''bc\phi'''$ there are two *distinct* residues $\widehat{\phi}_1 = \phi'$ and $\widehat{\phi}_2 = \phi'''$. Notice, indeed, that any path $\phi^* = \phi'\psi\phi'''$ cannot be a residue, as it would cross the !-node $!A \in \mathcal{G}(\mathcal{D}) \cap \mathcal{G}(\mathcal{D}')$, which is not crossed by ϕ.
  - The last case is where $\phi = \phi'x_iy_i\phi''abc\phi'''$, and so there is exactly one residue $\widehat{\phi} = \phi'u_iv_i\phi'''de\phi'''$. But then $\text{rk}_{\mathcal{D}'}(\widehat{\phi}) = \text{rk}_{\mathcal{D}}(\phi)$.

□

- The case where $r = $ c!p-vs-c!p is similar to the previous one. □

Proposition 43 (Invariance). *Let $\mathcal{D} \in \text{wrPLL}_2^\infty$ be a coderivation, and let $\mathcal{D} \rightarrow_{\text{cut}} \mathcal{D}'$ be a cut-elimination step reducing an exponential cut $r \neq$ c!p-vs-c!p. Then, any balanced exponential flow has at most one residue, and it is balanced.*

PROOF. The statement is straightforward if $\mathcal{D} \rightarrow_{\text{cut}} \mathcal{D}'$ is a commuting cut-elimination step or if it reduces a cut that is not crossed by ϕ. Indeed, notice that ϕ cannot be erased by the cut-elimination step, since ϕ has nesting level 0. So, let us suppose that the cut r reduced by $\mathcal{D} \rightarrow_{\text{cut}} \mathcal{D}'$ is crossed by ϕ. Notice that it cannot be the case that $r = $ c!p-vs-ax, so there are two cases:

- $r = $ c!p-vs-?w and ϕ crosses the active formulas of r with an edge b, as in the leftmost cut-elimination step of Figure 12. Since ϕ is balanced, every !-node crossed is a p-node, and so the nodes $!A$ are in the main branch of a nwb $\mathfrak{S}$. Since $p(\phi, \mathfrak{S}) > b(\phi[!A]) = 0$, if $?\Gamma$ is not empty then the exponential flow ϕ must be of the form $\phi'x_iy_i\phi''ab$, for some directed paths ϕ', ϕ''. In this case there is exactly one residue $\widehat{\phi} = \phi'$. Notice that if $?\Gamma$ were empty then there would be no residue.
- $r = $ c!p-vs-?b and ϕ crosses the active formulas of r with an edge b, as in the rightmost cut-elimination step of Figure 12. Since ϕ is balanced, every !-node crossed is a p-node, and so the nodes $!A$ are in the main branch of a nwb $\mathfrak{S}$. Moreover, since $p(\phi, \mathfrak{S}) > b(\phi[!A])$ the exponential flow ϕ must be of the form $\phi'x_iy_i\phi''abc\phi'''$, for some directed paths ϕ', ϕ'', ϕ''', and there is exactly one residue $\widehat{\phi} = \phi'u_iv_i\phi''de\phi'''$. □

C.2 Proofs of Subsection 5.3

Lemma 47. *Let $\mathcal{D} \in \text{wrPLL}_2^\infty$ with !-free conclusion. Then, the shallow cut-elimination strategy applied to $\mathcal{D}$ satisfies the following properties for every $1 \leq d \leq \mathbf{d}(\mathcal{D}) + 1$:*

(1) $\text{base}(\mathcal{D}^{d-1}) \rightarrow_{\text{cut}}^* \text{base}(\mathcal{D}_e^d)$.
(2) either $\mathbf{d}(\mathcal{D}^{d-1}) = 0$ or $\mathbf{d}(\mathcal{D}^d) = \mathbf{d}(\mathcal{D}^{d-1}) - 1$.
(3) $\lfloor \mathcal{D}_e^d \rfloor_{\text{rk}(\mathcal{D}_e^d)} \rightarrow_{\text{cut}}^* \text{base}(\mathcal{D}^d)$.

PROOF. Item 1 follows from the fact that shallow cuts that are not bordered only affect $\text{base}(\mathcal{D}^{d-1})$, so that $\text{base}(\mathcal{D}^{d-1}) \rightarrow_{\text{cut}}^* \text{base}(\mathcal{D}_e^d)$ by Proposition 27.

Let us prove Item 2 and Item 3, and let $\mathfrak{S}_1, \ldots, \mathfrak{S}_n$ ($n \geq 0$) be the nwbs of $\mathcal{D}_e^d$ (and of $\mathcal{D}^{d-1}$) with nesting level 0. By Proposition 68.1, since all shallow cuts are bordered, there are exactly n shallow cuts $r_1, \ldots, r_n$ and the active !-formula of each r_i, say $!A_i$, is principal for $\mathfrak{S}_i$. This means that all !-nodes of $\mathcal{G}(\mathcal{D}_e^d)$ are p-nodes So by Proposition 66.2 there is a (possibly repeating) list of balanced exponential flows $\phi_1, \ldots, \phi_n$ of $\mathcal{G}(\mathcal{D}_e^d)$ such that the node $!A_i$ is crossed by ϕ_i.

By Proposition 43, if we reduce a cut $r \neq$ c!p-vs-c!p crossed by a balanced exponential flow we obtain exactly one balanced exponential flow $\widehat{\phi}$. By Definition 44, $\widehat{\phi}$ crosses the residue $\widehat{r}$ of r, which is shallow and bordered by Remark 45.

Proposition 67.2 ensures that, as long as there are shallow bordered cuts in balanced exponential flows there are also cuts $r \neq$

c!p-vs-c!p crossed by them, so if **Phase 2** can only terminate whenever there is hereditarily no (shallow bordered) residue of $r_1, \ldots, r_n$, i.e., if the nwbs $\mathfrak{S}_1, \ldots, \mathfrak{S}_n$ are eventually erased by a c!p-vs-?w by Remark 45. Hence, if $n \geq 0$ then **Phase 2** decreases $\mathcal{D}_e^d$ by 1, and so $\mathbf{d}(\mathcal{D}^d) = \mathbf{d}(\mathcal{D}_e^d) - 1 = \mathbf{d}(\mathcal{D}^{d-1}) - 1$. This proves Item 2.

Finally, let $m \geq 0$ and $\mathcal{D}_e^d = \mathcal{D}_0 \to_{\text{cut}} \mathcal{D}_1 \to_{\text{cut}} \ldots \to_{\text{cut}} \mathcal{D}_m$ be the first m cut-elimination steps of **Phase 2** on $\mathcal{D}_e^d$. Since **Phase 2** only reduces shallow bordered cuts crossed by $\phi_1, \ldots, \phi_n$ and their (unique) residues, by Proposition 41 there are at most $\text{rk}_{\mathcal{D}}(\phi_i)$ $(\leq \text{rk}(\mathcal{D}_e^d))$ c!p-vs-?b steps involving a c!p rule in (the main branch of) $\mathfrak{S}_i$. Moreover, since c!p-vs-c!p is never reduced, only the first $\text{rk}(\mathcal{D}_e^d)$ c!p rules of $\mathfrak{S}_i$ (from bottom) are affected by **Phase 2**. This means that $\lfloor \mathcal{D}_e^d \rfloor_{\text{rk}(\mathcal{D}_e^d)} = \lfloor \mathcal{D}_0 \rfloor_{\text{rk}(\mathcal{D}_e^d)} \to_{\text{cut}} \mathcal{D}_1' \ldots \to_{\text{cut}} \mathcal{D}_m'$ for some finite approximations $(\mathcal{D}_i')_{1 \leq i \leq m}$ such that $\text{base}(\mathcal{D}_i') = \text{base}(\mathcal{D}_i)$. Moreover, since $\lfloor \mathcal{D}_e^d \rfloor_{\text{rk}(\mathcal{D}_e^d)}$ is a finite approximation, m is bounded by Proposition 27, and so $\lfloor \mathcal{D}_e^d \rfloor_{\text{rk}(\mathcal{D}_e^d)} \to_{\text{cut}}^* \mathcal{D}^*$, for some finite approximation $\mathcal{D}^*$ such thay $\text{base}(\mathcal{D}^*) = \text{base}(\mathcal{D}^d)$. By Item 2 it must be that $\mathbf{d}(\mathcal{D}^*) = 0$, and so $\mathcal{D}^* = \text{base}(\mathcal{D}^*)$. □

Proposition 68. *Let* $\mathcal{D} \in \text{pPLL}_2^\infty$ *be a coderivation of a !-free sequent. Then:*

(1) *If any cut of* $\mathcal{D}$ *with nesting level 0 is exponential, then any !-thread starts with the active formula of a cut.*

(2) *If* $\mathcal{D}$ *is cut-free, it is a derivation.*

PROOF. Item 1 is straightforward. As for Item 2. It suffices notice that if one of the premises of an inference rule $r \neq \text{cut}$ of pPLL_2^∞ contains a ! then so the conclusion of r does, recalling that instantiation in the $\exists$ rule requires !-freeness. As a consequence, $\mathcal{D}$ has no occurrence of ! and so it has no infinite branch by progressing condition of $\mathcal{D}$. By weak König lemma, $\mathcal{D}$ must be finite (hence, a derivation). □

THEOREM 48 (TERMINATION). *Let* $\mathcal{D} \in \text{wrPLL}_2^\infty$ *with !-free conclusion. Then, the shallow cut-elimination strategy applied to* $\mathcal{D}$ *terminates in a* finite *number of steps returning a cut-free derivation.*

PROOF. On the one hand, $\text{base}(\mathcal{D}^{d-1}) \to_{\text{cut}}^* \text{base}(\mathcal{D}_e^d)$ implies $\mathcal{D}^{d-1} \to_{\text{cut}}^{*m} \mathcal{D}_e^d$. On the other hand, $\lfloor \mathcal{D}_e^d \rfloor_{\text{rk}(\mathcal{D}_e^d)} \to_{\text{cut}}^* \mathcal{D}^d$ implies $\mathcal{D}_e^d \to_{\text{cut}}^{*e} \mathcal{D}^d$. Therefore, $\mathcal{D}^{d-1} \to_{\text{cut}}^{*r} \mathcal{D}^d$ for every $1 \leq d \leq \mathbf{d}(\mathcal{D}) + 1$. This means that every round is performed in a finite number of steps and, since there are only finitely many rounds, the strategy terminates. Moreover, by Item 2 we have $\mathbf{d}(\mathcal{D}^{\mathbf{d}(\mathcal{D})}) = 0$, and so no !-formula is in a sequent of $\mathcal{D}^{\mathbf{d}(\mathcal{D})}$ by Remark 17. By Proposition 68, this implies that $\mathcal{D}^{\mathbf{d}(\mathcal{D})}$ is a derivation (i.e., it is a finite coderivation). Therefore, by Proposition 27.3, $\mathcal{D}^{\mathbf{d}(\mathcal{D})+1}$ is a cut-free derivation. □

C.3 Proofs of Subsection 5.4

Lemma 50 (Polynomial modulus of continuity). *Let* $\mathcal{D} \in \text{wrPLL}_2^\infty$ *be a coderivation of a !-free sequent. Then, for some polynomial* $p : \mathbb{N} \to \mathbb{N}$ *depending solely on* $\mathbf{d}(\mathcal{D})$, $\lfloor\!\lfloor \mathcal{D} \rfloor\!\rfloor_{p(||\mathcal{D}||)}$ *rewrites by the shallow cut-elimination strategy to a cut-free* hyp-*free derivation.*

PROOF. By Lemma 49 we have:

$$||\mathcal{D}^d||_0 \in \quad O\left(\prod_{i=0}^{d} ||\mathcal{D}^0||_i^{6^{d+1-i}}\right) \quad \text{Lemma 49}$$
$$= \quad O\left(\prod_{i=0}^{\mathbf{d}(\mathcal{D})} ||\mathcal{D}^0||_i^{6^{\mathbf{d}(\mathcal{D})+1-i}}\right) \quad \text{Proposition 32}$$
$$= \quad O\left(||\mathcal{D}^0||^{\mathbf{d}(\mathcal{D}) \cdot 6^{\mathbf{d}(\mathcal{D})+1}}\right)$$
$$= \quad O\left(||\mathcal{D}||^{\mathbf{d}(\mathcal{D}) \cdot 6^{\mathbf{d}(\mathcal{D})+1}}\right)$$

Hence, since Proposition 27.2 implies $\text{rk}(\mathcal{D}_e^d) \leq |\text{base}(\mathcal{D}_e^d)| \in O(|\text{base}(\mathcal{D}^d)|) = O(||\mathcal{D}^d||_0)$, by Theorem 48.1,3 there is some $k > 0$ depending solely on $\mathbf{d}(\mathcal{D})$ and a constant $c > 0$ such that:

$$\lfloor\!\lfloor \mathcal{D}^{d-1} \rfloor\!\rfloor_{c \cdot ||\mathcal{D}||^k} \to_{\text{cut}}^* \lfloor\!\lfloor \mathcal{D}_e^d \rfloor\!\rfloor_{c \cdot ||\mathcal{D}||^k} \to_{\text{cut}}^* \lfloor\!\lfloor \mathcal{D}^d \rfloor\!\rfloor_{c \cdot ||\mathcal{D}||^k}$$

for any $0 \leq d \leq \mathbf{d}(\mathcal{D}) + 1$. Moreover, $\lfloor\!\lfloor \mathcal{D}^{\mathbf{d}(\mathcal{D})+1} \rfloor\!\rfloor_{c \cdot ||\mathcal{D}||^k} = \mathcal{D}^{\mathbf{d}(\mathcal{D})+1}$ by Theorem 48.2. Therefore, by Theorem 48 we have that $\lfloor\!\lfloor \mathcal{D} \rfloor\!\rfloor_{c \cdot ||\mathcal{D}||^k}$ rewrites by the shallow cut-elimination strategy to a cut-free hyp-free derivation. □

The following relation between the size of $\lfloor\!\lfloor \mathcal{D} \rfloor\!\rfloor_n$ and the cosize of $\mathcal{D}$ holds.

Proposition 69. *Let* $\mathcal{D} \in \text{wrPLL}_2^\infty$. *Then*

$$|\lfloor\!\lfloor \mathcal{D} \rfloor\!\rfloor_n| \in O(n^{\mathbf{d}(\mathcal{D})+1} \cdot ||\mathcal{D}||^{\mathbf{d}(\mathcal{D})+1}) \qquad .$$

PROOF. Let $\mathfrak{S}_i = \mathcal{D}_{v_i}$ with $v_i \in \text{border}(\mathcal{D})$. If $\mathbf{d}(\mathcal{D}) = 0$, then $|\lfloor\!\lfloor \mathcal{D} \rfloor\!\rfloor_n| = |\mathcal{D}| = ||\mathcal{D}||$. If $\mathbf{d}(\mathcal{D}) = d+1$, then $|\lfloor\!\lfloor \mathcal{D} \rfloor\!\rfloor_n| := |\text{base}(\mathcal{D})| + \sum_{i=1}^{k} \sum_{j=0}^{n} |\lfloor\!\lfloor \mathfrak{S}_i(j) \rfloor\!\rfloor_n|$ by definition. By the induction hypothesis $|\lfloor\!\lfloor \mathfrak{S}_i(j) \rfloor\!\rfloor_n| \in O(n^d \cdot ||\mathfrak{S}_i(j)||^d)$, hence $|\lfloor\!\lfloor \mathfrak{S}_i(j) \rfloor\!\rfloor_n| \in O(n^d \cdot ||\mathcal{D}||^d)$. Since $k \leq ||\mathcal{D}||$, then we have $|\lfloor\!\lfloor \mathcal{D} \rfloor\!\rfloor_n| \in O(||\mathcal{D}|| + n \cdot ||\mathcal{D}|| \cdot n^d \cdot ||\mathcal{D}||^d)$, and we conclude that $|\lfloor\!\lfloor \mathcal{D} \rfloor\!\rfloor_n| \in O(n^{d+1} \cdot ||\mathcal{D}||^{d+1})$. □

THEOREM 51. *[Soundness] Let* $f : (\{0,1\}^*)^n \to \{0,1\}^*$:
(1) *If* f *is representable in* wrPLL_2^∞ *then* $f \in \textbf{FP}/\textbf{poly}$;
(2) *If* f *is representable in* rPLL_2^∞ *then* $f \in \textbf{FP}$.

PROOF. We only show the case where f is unary for the sake of simplicity. Let $\underline{f} \in \text{wrPLL}_2^\infty$ represent f, and let us consider the following coderivation, with $s = b_1, \ldots, b_n \in \{0,1\}^*$:

$$\mathcal{D}_{f(s)} \quad := \quad \frac{\overbrace{f}\ \overbrace{s}}{\cfrac{\text{S}[\,] \multimap \text{S} \quad \text{S}[\,]}{\text{S}}\,{}_{-\circ_e}}$$

By Lemma 50 there are $\mathcal{D}_0, \mathcal{D}_1, \ldots, \mathcal{D}_m$ such that:

$$\lfloor\!\lfloor \mathcal{D}_{f(s)} \rfloor\!\rfloor_{c \cdot ||\mathcal{D}_{f(s)}||^k} = \mathcal{D}_0 \to_{\text{cut}} \mathcal{D}_1 \to_{\text{cut}} \ldots \to_{\text{cut}} \mathcal{D}_m = \underline{f(s)}$$

for some constant $c > 0$, and for some $k > 0$ depending solely on $\mathbf{d}(\mathcal{D}_{f(s)}) = \mathbf{d}(\underline{f})$ (since $\mathbf{d}(\underline{s}) = 0$). In particular, $||\mathcal{D}_{f(s)}|| \in O(||\underline{s}||) = O(|\underline{s}|) = O(|s|)$, where $|s|$ is the size of the string s. So, we have:

$$\lfloor\!\lfloor \mathcal{D}_{f(s)} \rfloor\!\rfloor_{c \cdot |s|^k} = \mathcal{D}_0 \to_{\text{cut}} \mathcal{D}_1 \to_{\text{cut}} \ldots \to_{\text{cut}} \mathcal{D}_m = \underline{f(s)}$$

for some constant $c > 0$ and some $k > 0$ depending solely on $\mathbf{d}(\underline{f})$. Moreover:

- By Proposition 69 we have

$$|\lfloor\!\lfloor \mathcal{D}_{f(s)} \rfloor\!\rfloor_{c \cdot |s|^k}| \in O(|s|^{k \cdot \mathbf{d}(\mathcal{D}_{f(s)})+1} \cdot ||\mathcal{D}_{f(s)}||^{\mathbf{d}(\mathcal{D}_{f(s)})+1})$$
$$= O(|s|^{k \cdot \mathbf{d}(\mathcal{D}_f)+1} \cdot |s|^{\mathbf{d}(\mathcal{D}_f)+1})$$
$$= O(|s|^{k \cdot \mathbf{d}(\underline{f})+1} \cdot |s|^{\mathbf{d}(\underline{f})+1}) = O(|s|^h)$$

for some $h > 0$ depending solely on $\mathbf{d}(f)$.

- By Proposition 27, we have $m \in O(|s|^{3h})$ and $|\mathcal{D}_i| \in O(|s|^{3h})$.

This means that we can construct a polysize family of circuits $C = (C_n)_{n\geq 0}$ such that, for any $n \geq 0$, on input $s = b_1, \ldots, b_n \in \{\mathbf{0}, \mathbf{1}\}^*$, $C_n(s)$ evaluates $\mathcal{D}_{f(s)}$ to $\underline{f(s)}$ and returns $f(s)$. Therefore, $f \in \mathbf{FP}/\mathrm{poly}$. Suppose now that $\underline{f}$ is representable in rPLL_2^∞. This means that $\underline{f}$ is regular, and so the function $n \mapsto C_n$ can be constructed uniformly by a polytime Turing machine. Therefore, $f \in \mathbf{FP}$. □

D PROOF OF SECTION 6

D.1 Proof of Proposition 54

We start with a structural property of the type systems that is straightforward consequence of the linearity restrictions introduced by the essential types.

Proposition 70. *If* $\mathcal{D} : \Gamma \vdash M : !\sigma$ *then* $\Gamma = !\Gamma'$ *and* $\mathcal{D}$ *is obtained from a typing derivation* $\mathcal{D}'$ *by one application of* $\mathsf{f!p}$, *followed by a series of applications of* $?\mathsf{w}$ *and* $?\mathsf{b}$.

PROOF. Straightforward, by induction on $\mathcal{D}$. □

We now introduce substitution properties for both types and typable terms.

Lemma 71. *If* $\Gamma \vdash M : \sigma$ *then* $\Gamma[\vec{C}/\vec{X}] \vdash M : \sigma[\vec{C}/\vec{X}]$ *for every* $\vec{C} = C_1, \ldots, C_n$ *and* $\vec{X} = X_1, \ldots, X_n$.

Lemma 72 (Substitution). *If* $\mathcal{D}_1 : \Gamma, x : \tau \vdash M : \sigma$ *and* $\mathcal{D}_2 : \Delta \vdash N : \tau$ *then there is a typing derivation* $S(\mathcal{D}_1, \mathcal{D}_2)$ *of* $\Gamma, \Delta \vdash M[N/x] : \sigma$.

PROOF. The proof is by induction on the lexicographic order over $(h(\mathcal{D}_1), s(\tau))$, where $h(\mathcal{D}_1)$ is the height of $\mathcal{D}_1$ and $s(\tau)$ is the number of symbols of the formula τ. The only interesting case is when $\mathcal{D}_1$ is obtained from $\mathcal{D}_1'$ by applying a $?\mathsf{b}$ rule, where $M = M'[x/y, x/z]$, $\tau = !\tau'$, and $\mathcal{D}_1'$ is

$$?\mathsf{b} \frac{\overset{\mathcal{D}_1'}{\Gamma, y : \tau', z : !\tau' \vdash M' : \sigma}}{\Gamma, x : !\tau' \vdash M'[x/y, x/z] : \sigma}$$

By Proposition 70 we have that $\Delta = !\Sigma$ and

$$\mathcal{D}_2 := {?\mathsf{b},?\mathsf{w}} \frac{\overset{\mathcal{D}_2'}{!\Sigma' \vdash N' : !\tau'}}{!\Sigma \vdash N : !\tau'} \qquad \mathcal{D}_2' := \mathsf{f!p} \frac{\overset{\mathcal{D}_2''}{\Sigma' \vdash N' : \tau'}}{!\Sigma' \vdash N' : !\tau'}$$

Since $h(\mathcal{D}_1') < h(\mathcal{D}_1)$ then $(h(\mathcal{D}_1'), s(!\tau)) < (h(\mathcal{D}_1), s(!\tau))$ and by induction hypothesis we have a typing derivation $S(\mathcal{D}_1', \mathcal{D}_2)$ of $\Gamma, !\Sigma' \vdash M'[N'/z] : \sigma$. Moreover, since $s(\tau) < s(!\tau)$ then $(h(S(\mathcal{D}_1', \mathcal{D}_2)), s(\tau)) < (h(\mathcal{D}_1), s(!\tau))$, and by applying the induction hypothesis again there is a typing derivation $S(S(\mathcal{D}_1', \mathcal{D}_2), \mathcal{D}_2')$ of $\Gamma, \Sigma', !\Sigma' \vdash M'[N'/z, N'/y] : \sigma$. We conclude by applying a series of $?\mathsf{b}$ rules and $?\mathsf{w}$ rules. □

Proposition 54 (Subject reduction). *Let* $\mathcal{D} : \Gamma \vdash M : \sigma$. *If* $M \to_\beta N$ *then there is* $\mathcal{D}'$ *such that* $\mathcal{D}' : \Gamma \vdash N : \sigma$.

PROOF. It suffices to check that the reduction rules given in Definition 52 preserve types. We consider the most interesting reduction rule, i.e., $M = (\lambda x.P)Q \to_\beta P[Q/x] = N$. By inspecting the typing rules in Figure 13, $\mathcal{D}$ must have the following structure:

$$\multimap_e \frac{\overset{\mathcal{D}_1}{\Sigma' \vdash \lambda x.P' : \tau \multimap B'} \quad \overset{\mathcal{D}_2}{\Delta' \vdash Q' : \tau}}{\Sigma', \Delta' \vdash (\lambda x.P')Q' : B'}$$
$$\delta \vdots$$
$$\overline{\Sigma, \Delta \vdash (\lambda x.P)Q : \sigma}$$

where:

- $\Gamma = \Sigma, \Delta$ and $\sigma = !.^n.!\forall\vec{X}.B$, for some $n \geq 0$, $\vec{X}$, and B.
- δ is a sequence of rules in $\{?\mathsf{w}, ?\mathsf{b}, \mathsf{f!p}, \forall_i, \forall_e\}$,
- $B = B'[\vec{C}/\vec{Y}]$, for some $\vec{C}$ and $\vec{Y}$ not free in Σ', Δ'
- $P'[\vec{x}/\vec{y}] = P$ and $Q'[\vec{x}/\vec{y}] = Q$, for some $\vec{x}, \vec{y}$.

By a similar reasoning, $\mathcal{D}_1$ has the following shape:

$$\multimap_i \frac{\overset{\mathcal{D}_1'}{\Sigma'', x : \tau' \vdash P'' : B''}}{\Sigma'' \vdash \lambda x.P'' : \tau' \multimap B''}$$
$$\varepsilon \vdots$$
$$\overline{\Sigma' \vdash \lambda x.P' : \tau \multimap B'}$$

where:

- ε is sequences of typing rules in $\{?\mathsf{w}, ?\mathsf{b}, \forall_i, \forall_e\}$,
- $B' = B''[\vec{D}/\vec{Z}]$ and $\tau = \tau'[\vec{D}/\vec{Z}]$, for some $\vec{D}$ and $\vec{Z}$ not free in Σ''
- $P''[\vec{z}/\vec{w}] = P'$ for some $\vec{z}, \vec{w}$.

Since $\tau = \tau'[\vec{D}/\vec{Z}]$, $B' = B''[\vec{D}/\vec{Z}]$ and $\vec{Z}$ do not occur free in Σ'', by Lemma 71 there is a typing derivation $\mathcal{D}_1''$ of $\Sigma'', x : \tau \vdash P'' : B'$. By Lemma 72 there is a typing derivation $S(\mathcal{D}_1'', \mathcal{D}_2)$ of $\Delta', \Sigma'' \vdash P''[Q'/x] : B'$. Finally, by applying the sequences of rules δ and ε we obtain:

$$\frac{\overset{S(\mathcal{D}_1'', \mathcal{D}_2)}{\Delta', \Sigma'' \vdash P''[Q'/x] : B'}}{\vdots}$$
$$\overline{\Delta', \Sigma' \vdash P'[Q'/x] : B'}$$
$$\vdots$$
$$\overline{\Sigma, \Delta \vdash P[Q/x] : \sigma}$$

□

D.2 Definability and data types in PTA_2 and nuPTA_2

Polymorphic type systems based on linear logic typically encode inductive datatypes by universally quantified types (see, e.g., [28]). Examples are natural numbers, defined by $\mathbf{N} := !(X \multimap X) \multimap X \multimap X$. Because of linearity restrictions on polymorphism, however, parsimonious logic cannot freely apply instantiation when encoding functions over inductive datatypes. As a consequence,

its computational strength relative to standard notions of representability [6] would be fairly poor. To circumvent this technical issue, following previous works on parsimonious logic [39, 40], we adopt a parametric notion of representability, where natural numbers are defined by types of the form $!(A \multimap A) \multimap A \multimap A$, i.e., by instantiations of $\mathbf{N}$.

To this end, we generalise the usual notion of lambda definability [6] to different kinds of input data:

Definition 73 (Representability [28]). *Let* $f : \mathbb{I}_1 \times \ldots \times \mathbb{I}_n \to \mathbb{O}$ *be a total function and let the elements* $o \in \mathbb{O}$ *and* $i_j \in \mathbb{I}_j$ *for* $0 \leq j \leq n$ *be encoded by terms* $\underline{o}$ *and* $\underline{i_j}$ *such that* $\vdash \underline{o} : \mathbf{O}$ *and* $\vdash \underline{i_j} : \mathbf{I}_j$. *Then,* f *is representable in* nuPTA$_2$ *(resp.* PTA$_2$*) if there is a term* $\underline{f} \in \Lambda_{\text{stream}}$ *such that* $\vdash \underline{f} : \mathbf{I}_1 \multimap \ldots \multimap \mathbf{I}_n \multimap \mathbf{O}$ *in* nuPTA$_2$ *(resp.* $\overline{\text{PTA}}_2$*) and*

$$f \, i_1 \ldots i_n = o \quad \Longleftrightarrow \quad \underline{f} \, \underline{i_1} \ldots \underline{i_n} \to_\beta^* \underline{o}$$

We adopt the usual notational convention $M^n N$ $(n \geq 0)$ defined inductively as $M^0(N) := N$ and $M^{n+1}(N) := M(M^n(N))$. We also set $M \circ N := \lambda z.M(Nz)$, which generalises to the n-ary case $M_1 \circ \ldots \circ M_n := \lambda z.M_1(M_2(\ldots (M_n z)))$. Finally, the n-ary tensor product (with $n \geq 3$) can be defined from the binary one by setting $M_1 \otimes \ldots \otimes M_n := (M_1 \otimes \ldots \otimes M_{n-1}) \otimes M_n$, $\sigma_1 \otimes \ldots \otimes \sigma_n := (\sigma_1 \otimes \ldots \otimes \sigma_{n-1}) \otimes \sigma_n$, and let $x_1 \otimes \ldots \otimes x_n = z$ in $M := $ let $y \otimes x_n = z$ in (let $x_1 \otimes \ldots \otimes x_{n-1} = y$ in M). We also use the shorthand notation $\sigma^n := \sigma \otimes \overset{n}{\ldots} \otimes \sigma$.

In what follows we encode some relevant data types and their basic operations in nuPTA$_2$ (and PTA$_2$).

Definition 74 (Booleans). Booleans $\mathbf{0}, \mathbf{1}$ and basic Boolean operations are encoded as in Figure 14. The can be typed by $\mathbf{B} := \forall X.(X \otimes X) \multimap (X \otimes X)$.

The following is a straightforward consequence of the encodings in Figure 14.

Proposition 75 (Functional completeness). *Every Boolean function* $f : \{\mathbf{0}, \mathbf{1}\}^n \to \{\mathbf{0}, \mathbf{1}\}^m$ *with* $n \geq 0$, $m > 0$ *can be represented by a term* $\underline{f} \in \Lambda_{\text{stream}}$ *such that* $\vdash \underline{f} : \mathbf{B}^n \multimap \mathbf{B}^m$.

Notice that Proposition 75 crucially relies on the terms $\mathbf{C_B}$ and $\mathbf{W_B}$, which duplicate and erase Booleans in a purely linear fashion. Following [37], we can generalise linear erasure of data to a fairly large class of types.

Definition 76 (Π_1 and $e\Pi_1$ types [37]). Let A be a type build from $\mathbf{1}, \otimes, \multimap, \forall$. We say that A is $e\Pi_1$ if every $\forall$-type occurring in it is inhabited.

Proposition 77 (Linear erasure [15, 37]). *For any closed type* A *in* $e\Pi_1$ *there is a term* $\mathbf{W}_A$ *in a term that inhabits* $A \multimap \mathbf{1}$.

Proposition 78 (Conditional). *For any* A *in* $e\Pi_1$, *the following rule is derivable:*

$$\text{cond} \, \frac{\vdash R : A \quad \vdash L : A}{x : \mathbf{B} \vdash \text{if } x \text{ then } R \text{ else } L : A}$$

where if x then R else L *satisfies the following reductions:*

$$\text{if } \underline{\mathbf{1}} \text{ then } R \text{ else } L \to_\beta^* R$$
$$\text{if } \underline{\mathbf{0}} \text{ then } R \text{ else } L \to_\beta^* L$$

PROOF. We set if x then R else $L := \pi_1^2(xRL)$, where π_1^2 is as in Figure 14. □

Definition 79 (Streams of Booleans). A stream (of Booleans) α is encoded by a term $\mathbf{M}$ such that $\mathbf{M}(i) := \underline{\alpha(i)}$. We write $\underline{\alpha}$ for the encoding of α. Streams can be typed by $\mathbf{Stream} := \omega\mathbf{B}$.

Definition 80 (Natural numbers and Boolean strings). The encoding of Boolean strings and natural numbers is as follows, for any $n \geq 0$ and $s = b_1 \cdots b_n \in \{\mathbf{0}, \mathbf{1}\}^*$:

$$\underline{n} := \lambda f.\lambda z.f^n z$$
$$\underline{s} := \lambda f.\lambda z.f \, \underline{b_n}(f \, \underline{b_{n-1}}(\ldots (f \, \underline{b_1} \, z) \ldots))$$

For any type A, natural numbers and Boolean strings can be typed, respectively, by

$$\mathbf{N}[A] := !(A \multimap A) \multimap A \multimap A$$
$$\mathbf{S}[A] := !(\mathbf{B} \multimap A \multimap A) \multimap A \multimap A$$

With $\mathbf{N}[]$ we denote $\mathbf{N}[A]$ for some A, and similarly for $\mathbf{S}[]$.

We need to encode the function that, when applied to a Boolean string, returns its length:

Proposition 81 (Length). *There exists a term* length *of type* $\mathbf{S}[A] \multimap \mathbf{N}[A]$ *satisfying the following reduction, for all* $s = b_1 \cdots b_n \in \{\mathbf{0}, \mathbf{1}\}^*$:

$$\text{length} \, \underline{s} \to_\beta^* \underline{n}$$

PROOF. We set length $:= \lambda s.\lambda f.s(\lambda x.\lambda y.\text{let } \mathsf{l} = \mathbf{W_B} \, x \text{ in } fy)$, where $\mathbf{W_B}$ is as in Figure 14. □

The following proposition shows that encodings of natural numbers and Boolean strings can be used as iterators.

Proposition 82 (Iteration). *For any* A, *the following rule is derivable*

$$\text{iter}_\mathbf{N} \, \frac{!\Gamma \vdash S : !(A \multimap A) \quad \Delta \vdash B : A}{!\Gamma, \Delta, n : \mathbf{N}[A] \vdash \text{iter}_\mathbf{N} \, n \, S \, B : A}$$

where $\text{iter}_\mathbf{N} \, n \, S \, B$ *satisfies the reduction*

$$\text{iter}_\mathbf{N} \, \underline{n} \, S \, B \quad \to_\beta^* \quad S^n B$$

Similarly, the following rule is derivable:

$$\text{iter}_\mathbf{S} \, \frac{!\Gamma \vdash S_0 : !(A \multimap A) \quad !\Gamma \vdash S_1 : !(A \multimap A) \quad \Delta \vdash B : A}{!!\Gamma, \Delta, n : \mathbf{S}[A] \vdash \text{iter}_\mathbf{S} \, n \, S_0 \, S_1 : A}$$

where $\text{iter}_\mathbf{S} \, n \, S_0 \, S_1$ *satisfies the reduction*

$$\text{iter}_\mathbf{N} \, \underline{b_1 \cdots b_n} \, S_0 \, S_1 B \quad \to_\beta^* \quad S_{b_n} \ldots S_{b_1} B$$

PROOF. It suffices to set, respectively, $\text{iter}_\mathbf{N} := \lambda \, n.\lambda s.\lambda b.nsb$ and $\text{iter}_\mathbf{S} := \lambda \, s\lambda t.\lambda u.\lambda b.stub$. □

Our next goal is to show that any polynomial over natural numbers can be encoded in nuPTA$_2$ (and PTA$_2$). The encoding of polynomials requires nesting types, so we introduce a notation for denoting iterated nesting in a succinct way.

Definition 83 (Nesting). Let A be a type. We define $\mathbf{N}_A[d]$ and $\mathbf{S}_A[d]$ by induction on $d \geq 0$:

$$\mathbf{N}_A[0] := A \qquad\qquad \mathbf{S}_A[0] := A$$
$$\mathbf{N}_A[d+1] := \mathbf{N}_A[\mathbf{N}_A[d]] \qquad \mathbf{S}_A[d+1] := \mathbf{S}_A[\mathbf{S}_A[d]]$$

If A is clear from the context, we simply write $\mathbf{N}[d]$ and $\mathbf{S}[d]$.

$$
\begin{array}{rll}
\underline{1} & := & \lambda x.\lambda y.x \otimes y & : \mathbf{B} \\
\underline{0} & := & \lambda x.\lambda y.y \otimes x & : \mathbf{B} \\
\mathbf{W_B} & := & \lambda b.\text{let } x_1 \otimes x_2 = b(\mathsf{I} \otimes \mathsf{I}) \text{ in let } \mathsf{I} = x_2 \text{ in } x_1 & : \mathbf{B} \multimap \mathbf{1} \\
\pi_1^2 & := & \lambda x.\text{let } x_1 \otimes x_2 = x \text{ in let } \mathsf{I} = \mathbf{W_B}\, x_2 \text{ in } x_1 & : \mathbf{B} \otimes \mathbf{B} \multimap \mathbf{B} \\
\mathbf{C_B} & := & \lambda b.\pi_1^2(b(\underline{1} \otimes \underline{1}) \otimes (\underline{0} \otimes \underline{0})) & : \mathbf{B} \multimap \mathbf{B} \otimes \mathbf{B} \\
\underline{\neg} & := & \lambda b.\lambda x.\lambda y.b(y \otimes x) & : \mathbf{B} \multimap \mathbf{B} \\
\underline{\vee} & := & \lambda b_1.\lambda b_2.\pi_1^2(b_1 \underline{0}\, b_2) & : \mathbf{B} \otimes \mathbf{B} \multimap \mathbf{B}
\end{array}
$$

Figure 14: Encoding of basic operations on Booleans.

Proposition 84. *For any $d \geq 0$, there exist a term $\text{down}_{\mathsf{N}}^d$ of type $\mathsf{N}[d+1] \multimap \mathsf{N}[d]$ and a term $\text{down}_{\mathsf{S}}^d$ of type $\mathsf{S}[d+1] \multimap \mathsf{S}[d]$ satisfying the following reductions:*

$$
\begin{array}{rcl}
\text{down}_{\mathsf{N}}^d\, \underline{n} & \to_\beta^* & \underline{n} \\
\text{down}_{\mathsf{S}}^d\, \underline{n} & \to_\beta^* & \underline{n}
\end{array}
$$

Proof. Straightforward, by setting $\text{down}_{\mathsf{N}}^d := \lambda x.\text{iter}_{\mathsf{N}}\, x\, \text{succ}\, \underline{0}$ and $\text{down}_{\mathsf{S}}^d := \lambda x.\text{iter}_{\mathsf{N}}\, x\, (\lambda b.\lambda s.\lambda c.\lambda z.cb(scz))\underline{\epsilon}$. $\square$

Definition 85 (Successor, addition, multiplication). Successor, addition and multiplication can be represented by the following terms:

$$
\begin{array}{rcl}
\text{succ} & := & \lambda n.\lambda f.\lambda z.n(f)(fz) \\
\text{add} & := & \lambda n.\lambda m.\text{iter}_{\mathsf{N}}\, n\, (\text{succ})\, m \\
\text{mult} & := & \lambda n.\lambda m.\text{iter}_{\mathsf{N}}\, m\, (\lambda y.\text{add}\, n\, y)\, \underline{0}
\end{array}
$$

they are typable as follows:

$$
\begin{array}{l}
\vdash \text{succ} : \mathsf{N}[i] \multimap \mathsf{N}[i] \\
\vdash \text{add} : \mathsf{N}[i+1] \multimap \mathsf{N}[i] \multimap \mathsf{N}[i] \\
\vdash \text{mult} : !\mathsf{N}[i+1] \multimap \mathsf{N}[i+1] \multimap \mathsf{N}[i]
\end{array}
$$

Theorem 86 (Polynomial completeness). *Let $p(x) : \mathbb{N} \to \mathbb{N}$ be a polynomial with degree $\delta(p) > 0$. Then there is a term $\underline{p}$ representing p such that, for any $i \geq 0$:*

$$
x : !^{\delta(p)-1}\mathsf{N}[\delta(p)+i] \vdash \underline{p} : \mathsf{N}[i]
$$

Proof. For the sake of readability, we will avoid writing the index i in typing judgements. Thus, with $\mathsf{N}[n]$ we mean $\mathsf{N}[n+i]$.

Consider a polynomial $p(x) : \mathbb{N} \to \mathbb{N}$ in Horner normal form, i.e., $p(x) = a_0 + x(a_1 + x(\ldots(a_{n-1} + xa_n)\ldots))$. We actually show something stronger:

$$
x_0 : \mathsf{N}[1], x_1 : !\mathsf{N}[2], \ldots, x_{n-1} : !^{n-1}\mathsf{N}[n] \vdash \hat{p} : \mathsf{N}[0] \qquad (4)
$$

where $\hat{p} = a_0 + x_0(a_1 + x_1(\ldots(a_{n-1} + x_{n-1}a_n)\ldots))$. The proof is by induction on $\delta(p) = n$. If $\delta(p) = 1$ then $\hat{p} = a_0 + x_0 a_1$, and we simply set $\hat{p} := \text{add}\, \underline{a_0}\, (\text{mult}\, \underline{a_1}\, x_0)$. If $\delta(p) > 1$ then $\hat{p} = a_0 + x_0 \hat{q}$ with $\hat{q} := a_1 + x_1(a_2 + x_2(\ldots(a_{n-1} + x_{n-1}a_n)\ldots))$. By induction hypothesis on q we have

$$
x_1 : \mathsf{N}[1], x_2 : !\mathsf{N}[2], \ldots, x_{n-1} : !^{n-2}\mathsf{N}[n-1] \vdash \hat{q} : \mathsf{N}[0]
$$

By repeatedly applying $\text{down}_{\mathsf{N}}^k$ for appropriate k we obtain a term M such that:

$$
x_1 : \mathsf{N}[2], x_2 : !\mathsf{N}[3], \ldots, x_{n-1} : !^{n-2}\mathsf{N}[n] \vdash M : \mathsf{N}[0]
$$

We set $\hat{p} := \text{add}\, \underline{a_0}\, (\text{mult}\, M\, x_0)$, which is typable as:

$$
x_0 : \mathsf{N}[1], x_1 : !\mathsf{N}[2], \ldots, x_{n-1} : !^{n-1}\mathsf{N}[n] \vdash \hat{p} : \mathsf{N}[0]
$$

and we can conclude since $\delta(q) = \delta(p) - 1$.

Now, to prove the theorem it suffices to repeatedly apply $\text{down}_{\mathsf{N}}^k$ for appropriate k to the typable term in Equation (4) in order to get a term N that represents $\hat{p}$ and typable as

$$
x_0 : \mathsf{N}[n], x_1 : !\mathsf{N}[n], \ldots, x_{n-1} : !^{n-1}\mathsf{N}[n] \vdash \hat{p} : \mathsf{N}[0]
$$

By applying a series of ?b we obtain a term $\underline{p}$ representing the polynomial p and such that $x : !^{\delta(p)-1}\mathsf{N}[\delta(p)] \vdash \underline{p} : \mathsf{N}[0]$. $\square$

D.3 Encoding polytime Turing machines with polynomial advice in nuPTA$_2$

In this subsection we show how to encode a Turing machine working in polynomial time with access to advice in nuPTA$_2$, following essentially [28, 37].

W.l.o.g., we will assume that the alphabet of the machine is composed by the two symbols $\mathbf{1}$ and $\mathbf{0}$[7], and that final states are divided into accepting and rejecting.

A *configuration C* of the machine will be represented by a term of the following form:

$$
\underline{C} := \lambda c.(cb_{\underline{0}}^l \circ \ldots \circ cb_{\underline{n}}^l) \otimes (cb_{\underline{0}}^r \circ \ldots \circ cb_{\underline{m}}^r) \otimes \underline{q} \otimes \underline{\alpha} \qquad (5)
$$

where:

- $b_0^r \in \{0, 1\}$ is the scanned symbol
- $b_1^r \cdots b_m^r \in \{0, 1\}$ are the symbols of the tape to the right of the scanned symbol
- $b_n^l \cdots b_0^l \in \{0, 1\}$ are the symbols of the tape to the left of the scanned symbol (notice that we encode this tuple in reverse order)
- $q = b_1 \cdots b_k \in \{0, 1\}$ is the (encoding of the) current state of the machine.
- α represents the advice of the machine as a single Boolean stream (see Proposition 6).

Terms as in Equation (5) have the following type:

$$
\mathbf{TM} := \forall X.!(\mathbf{B} \multimap X \multimap X) \multimap ((X \multimap X)^2 \otimes \mathbf{B}^k \otimes \mathbf{Stream})
$$

where **Stream** is as in Definition 79.

The *initial configuration C_0* describes a machine with tape filled by blank symbols (here **00**s) the head at the beginning of the tape

---

[7]We can clearly encode the alphabet $\Sigma = \{\mathbf{0}, \mathbf{1}, \sqcup\}$ within the alphabet $\{\mathbf{0}, \mathbf{1}\}$, by setting $\sqcup := \mathbf{00}$, $\mathbf{0} := \mathbf{01}$ and $\mathbf{1} := \mathbf{11}$.

and in the initial state q_0. To render the construction of the initial configuration in nuPTA$_2$, we define the following term:

$$\text{init} := \lambda n.\lambda c.(\lambda z.z) \otimes n(\lambda z'.c\underline{0}(c\underline{0}z')) \otimes \underline{q_0} \otimes \underline{\alpha} \qquad (6)$$

It takes the encoding of a natural number n in input and returns the term

$$\underline{C_0} := \lambda c.(\lambda z.z) \otimes (c\underline{0} \circ \overset{?n}{\ldots} \circ c\underline{0}) \otimes \underline{q} \otimes \underline{\alpha}$$

representing the first n blank symbols of the tape. Terms as in Equation (6) have the type below

$$\mathbf{N}[d] \multimap \mathbf{Stream} \multimap \mathbf{TM}$$

for all $d \geq 0$.

Following [28, 37], in order to show that Turing machine transitions are representable we consider two distinct phases:

- A *decomposition phase*, where the encoding of the configuration C is decomposed to extract the symbols b_0^l, b_0^r.
- A *composition phase*, where the components of C are assembled back to get the configuration of the machine after the transition.

The decomposition of a configuration has type **ID**:

$$\mathbf{ID} := \forall X.!(\mathbf{B} \multimap A[X]) \multimap (A[X]^2 \otimes B[X]^2 \otimes \mathbf{B}^k \otimes \mathbf{Stream})$$

where $A[X] := X \multimap X$ and $B[X] := (\mathbf{B} \multimap A[X]) \otimes \mathbf{B}$.

The decomposition phase is described by the term dec of type $\mathbf{TM} \multimap \mathbf{ID}$ defined as follows:

$$\text{dec} := \lambda m.\lambda c.\text{let } l \otimes r \otimes q \otimes \alpha = m(F[c]) \text{ in}$$
$$(\text{let } s_l \otimes c_l \otimes b_0^l = l(\mathsf{I} \otimes (\lambda x.\text{let } \mathsf{I} = \mathbf{W_B}\, x \text{ in } \mathsf{I}) \otimes \underline{0}) \text{ in}$$
$$(\text{let } s_r \otimes c_r \otimes b_0^r = r(\mathsf{I} \otimes (\lambda x.\text{let } \mathsf{I} = \mathbf{W_B}\, x \text{ in } \mathsf{I}) \otimes \underline{0}) \text{ in}$$
$$s_l \otimes s_r \otimes c_l \otimes \otimes b_0^l \otimes c_r \otimes b_0^r \otimes q \otimes \underline{\alpha})) \quad (7)$$

where $\mathbf{W_B}$ is the eraser for **B** given by Proposition 77, and $F[x] := \lambda b.\lambda z.\text{let } g \otimes h \otimes i = z \text{ in } (h\, i \circ g) \otimes x \otimes b$, which is typable as:

$$x : \mathbf{B} \multimap A[X] \vdash F[x] : B[(A[X] \otimes B[X])/X]$$

The term dec in Equation (7) satisfies the reduction in Figure 15.

Analogously, the composition phase is described by the term comp of type $\mathbf{ID} \multimap \mathbf{TM}$ defined as follows:

$$\text{comp} := \lambda s.\lambda c.\text{let } l \otimes r \otimes c_l \otimes b_l \otimes c_r \otimes b_r \otimes q \otimes \alpha = s\, c \text{ in}$$
$$\text{let } h \otimes t = \text{pop } \alpha \text{ in } (\text{let } b' \otimes q' \otimes m = \underline{\delta}(b_r \otimes h \otimes q) \text{ in}$$
$$((\text{if } m \text{ then } R \text{ else } L)b'q'(l \otimes r \otimes c_l \otimes b_l \otimes c_r) \otimes t) \quad (8)$$

where if m then R else L is defined as in Proposition 78, and

$$R := \lambda b'.\lambda q'.\lambda s.\text{let } l \otimes r \otimes c_l \otimes b_l \otimes c_r = s \text{ in}$$
$$(c_r b' \circ c_l b_l \circ l) \otimes r \otimes q'$$

$$L := \lambda b'.\lambda q'.\lambda s.\text{let } l \otimes r \otimes c_l \otimes b_l \otimes c_r = s \text{ in}$$
$$l \otimes (c_l b_l \circ c_r b' \circ r) \otimes q'$$

The term comp in Equation (8) satisfies the reduction in Figure 15, where $\delta : \{0,1\}^{k+2} \to \{0,1\}^{k+3}$ is the transition function of the Turing machine, which takes as an extra input the first bit of the current advice stack, i.e., the head of the stream α.

Remark 87. Notice that in comp the variable m has type **B** and is applied to the terms R and L. This requires to apply to the variable m the rule $\forall_e$, which instantiates the type variable X with the !-free type $\mathbf{B} \multimap \mathbf{B}^k \multimap ((X \multimap X)^2 \otimes (\mathbf{B} \multimap X \multimap X) \otimes \mathbf{B} \otimes (\mathbf{B} \multimap X \multimap X)) \multimap (X \multimap X)^2 \otimes \mathbf{B}^k$.

By combining the above terms we obtain the encoding of the Turing machine transition step:

$$\text{Tr} := \text{comp} \circ \text{dec} \qquad (9)$$

with type $\mathbf{TM} \multimap \mathbf{TM}$.

We now need a term that encodes the *initialisation* of the machine with an input Boolean string. This is given by the term In of type $\mathbf{S}[\mathbf{TM}] \multimap \mathbf{TM} \multimap \mathbf{TM}$ defined as follows:

$$\text{In} := \lambda s.\lambda m.s(\lambda b.(Tb) \circ \text{dec})\, m \qquad (10)$$

where dec is defined as in Equation (7) and

$$T := \lambda b.\lambda s.\lambda c.\text{let } l \otimes r \otimes c_l \otimes b_l \otimes c_r \otimes b_r \otimes q \otimes \alpha = sc \text{ in}$$
$$(\text{let } \mathsf{I} = \mathbf{W_B}\, b_r \text{ in } (Rbq(l \otimes r \otimes c_l \otimes b_l \otimes c_r)) \otimes \alpha)$$

$$R := \lambda b'.\lambda q'.\lambda s.\text{let } l \otimes r \otimes c_l \otimes b_l \otimes c_r = s \text{ in}$$
$$((c_r b' \circ c_l b_l \circ l) \otimes r \otimes q')$$

Intuitively, the term In defines a function that, when supplied with a Boolean string s and a Turing machine M, writes s as input on the tape of M.

Finally, we need a term that *extracts* the output string from the final configuration. This is given by the term Ext of type $\mathbf{TM} \multimap \mathbf{S}$, defined as follows:

$$\text{Ext} := \lambda s.\lambda c.\text{let } l \otimes r \otimes q \otimes \alpha = sc \text{ in}$$
$$\text{let } \mathsf{I} = \mathbf{W}_{\mathbf{B}^{k+1}}(q \otimes (\text{disc } \alpha)) \text{ in } l \circ r \qquad (11)$$

where disc is the eraser for streams (see Figure 13) and $\mathbf{W}_{\mathbf{B}^{k+1}}$ is the eraser for $\mathbf{B}^{k+1}$ given by Proposition 77.

We can now prove our fundamental theorem:

THEOREM 56. *Let $f : (\{0,1\}^*)^n \to \{0,1\}^*$:*
(1) If $f \in \mathbf{FP}/\text{poly}$ then f is representable in nuPTA$_2$;
(2) If $f \in \mathbf{FP}$ then f is representable in PTA$_2$.

PROOF. We only show the case where f is a unary function for the sake of simplicity. Let us prove Item 1. If $f \in \mathbf{FP}/\text{poly}$ then, $f \in \mathbf{FP}(\mathbb{R})$ by Proposition 6, so there is a polynomial Turing machine computing f that performs polynomially many queries to bits of a Boolean stream α. Let $p(x)$ and $q(x)$ be polynomials bounding, respectively, the time and space of the Turing machine, and let $\delta(p) = m$ and $\delta(q) = l$ be their degrees. By Theorem 86 we obtain $\underline{p}$ and $\underline{q}$ typable as:

$$y : !^{m-1}\mathbf{N}[m+1] \vdash \underline{p} : \mathbf{N}[1]$$
$$z : !^{l-1}\mathbf{N}[l+1] \vdash \underline{q} : \mathbf{N}[1]$$

where $\mathbf{N}[i]$ is shorthand notation for $\mathbf{N}_{\mathbf{TM}}[i]$ (see Definition 83). By applying Lemma 72 and Proposition 81, we have:

$$s' : !^{m-1}\mathbf{S}[m+1] \vdash P : \mathbf{N}[1]$$
$$s'' : !^{l-1}\mathbf{S}[l+1] \vdash Q : \mathbf{N}[1]$$

$$\text{dec } (\lambda c.(cb^l_{\underline{0}} \circ \ldots \circ cb^l_{\underline{n}}) \otimes (cb^r_{\underline{0}} \circ \ldots \circ cb^r_{\underline{m}}) \otimes \underline{q} \otimes \underline{\alpha}) \;\to^*_\beta\; \lambda c.(cb^l_1 \circ \ldots \circ cb^l_n) \otimes (cb^r_1 \circ \ldots \circ cb^r_m) \otimes c \otimes b^l_0 \otimes c \otimes b^r_0 \otimes \underline{q} \otimes \underline{\alpha}$$

$$\text{comp } (\lambda c.(cb^l_1 \circ \ldots \circ cb^l_n) \otimes (cb^r_1 \circ \ldots \circ cb^r_m) \otimes c \otimes b^l_0 \otimes c \otimes b^r_0 \otimes \underline{q} \otimes \underline{\alpha})$$

$$\to^*_\beta \begin{cases} \lambda c.(cb' \circ cb^l_0 \circ \ldots \circ cb^l_n) \otimes (cb^r_1 \circ \ldots \circ cb^r_m) \otimes \underline{q'} \otimes \underline{tl(\alpha)} & \text{if } \delta(b^r_0, hd(\alpha), q) = (b', q', \text{Right}) \\ \lambda c.(cb^l_1 \circ \ldots \circ cb^l_n) \otimes (cb^l_0 \circ cb' \circ cb^r_1 \circ \ldots \circ cb^r_m) \otimes q' \otimes \underline{tl(\alpha)} & \text{if } \delta(b^r_0, hd(\alpha), q) = (b', q', \text{Left}) \end{cases}$$

Figure 15: Reductions for dec **and** comp**.**

where $P := p[\text{length } s'/y]$: and $Q := q[\text{length } s''/z]$. On the other hand, by applying again Lemma 72 and Equations (6) and (9) to (11):

$$t : S[1], p : N[1], q : N[1] \vdash \text{Ext}((p\,\text{Tr})(\ln t\,(\text{init } q\,\alpha))) : S$$

By putting everything together we have:

$$s' : !^{m-1}S[m+1], s'' : !^{l-1}S[l+1], t : S[1] \vdash N : S$$

where $N := \text{Ext}((P\,\text{Tr})(\ln t\,(\text{init } Q\,\alpha)))$. By repeatedly applying ?b and down^k_S for appropriate k we obtain a term M representing f such that:

$$s : !^{\max(m,l)}S[\max(m,l)+1] \vdash M : S$$

By applying down^1_S we obtain

$$x : S[!^{\max(m,l)}S[\max(m,l)+1]] \vdash M[\text{down}^1_S x/s] : S$$

We set $\underline{f} := M[\text{down}^1_S x/s]$, so that $x : S[] \vdash \underline{f} : S$.

Point Item 2 follows directly from Item 1 by stripping away streams from the above encoding. □

D.4 Proof of Theorem 57

Definition 88 (Translation). We define a translation $(\_)^\dagger$ from nuPTA$_2$ to nuPLL$_2$ mapping typing derivations of nuPTA$_2$ to derivations of nuPLL$_2$ such that, when restricted to typing derivations of PTA$_2$, it returns derivations of PLL$_2$:

- It maps types of nuPTA$_2$ to formulas of nuPLL$_2$ according to the following inductive definition:

$$\begin{aligned} X^\dagger &:= X \\ 1^\dagger &:= 1 \\ (\sigma \multimap A)^\dagger &:= \sigma^\dagger \multimap A^\dagger \\ (\forall X.A)^\dagger &:= \forall X.A^\dagger \\ (\sigma \otimes \tau)^\dagger &:= \sigma^\dagger \otimes \tau^\dagger \\ (!\sigma)^\dagger &:= !\sigma^\dagger \\ (\omega\sigma)^\dagger &:= !\sigma^\dagger \end{aligned}$$

we notice that $\sigma^\dagger[\tau^\dagger/X] = (\sigma[\tau/X])^\dagger$.
- It maps a context $\Gamma = x_1 : \sigma_1, \ldots, x_n : \sigma_n$ to a sequent $\Gamma^\dagger = \sigma_1^\dagger, \ldots, \sigma_n^\dagger$.
- It maps judgements $\Gamma \vdash M : \tau$ to sequents $\Gamma^{\dagger\perp}, \tau^\dagger$.
- It maps a typing rule to gadgets as in Figure 16 and Figure 17.

The two lemmas below represent stronger versions of Lemma 72 and Proposition 54, respectively.

Lemma 89. *For any $\mathcal{D}_1 : \Gamma \vdash M : \sigma$ and $\mathcal{D}_2 : \Delta \vdash N : \tau$ there is $S(\mathcal{D}_1, \mathcal{D}_2)$ such that:*

$$\text{cut}\frac{\left(\overbrace{\Delta \vdash N : \tau}^{\mathcal{D}_1}\right)^\dagger \left(\overbrace{\Gamma, x : \tau \vdash M : \sigma}^{\mathcal{D}_2}\right)^\dagger}{\Gamma^{\dagger\perp}, \Delta^{\dagger\perp}, \sigma^\dagger} \to^*_{\text{cut}} \left(\overbrace{S(\mathcal{D}_1, \mathcal{D}_2)}{\Gamma, \Delta \vdash M[N/x] : \sigma}\right)^\dagger$$

PROOF. It suffices to check that the derivation $S(\mathcal{D}_1, \mathcal{D}_2)$ can be stepwise computed by the cut-elimination rules. □

Lemma 90. *Let $\mathcal{D}_1 : \Gamma \vdash M_1 : \sigma$. If $M_1 \to_\beta M_2$ then there is a typing derivation $\mathcal{D}_2 : \Gamma \vdash M_2 : \sigma$ such that $\mathcal{D}_1^\dagger \to^*_{\text{cut}} \mathcal{D}_2^\dagger$.*

PROOF. It suffices to check the statement for the reduction rules in Definition 52, by inspecting the cut-elimination rules of nuPLL$_2$. We consider the two most relevant cases. If $M_1 = \text{pop } \mathbf{M}$ and $M_2 = hd(\mathbf{M}) \otimes tl(\mathbf{M})$, then w.l.o.g. $\mathcal{D}_1$ has the following shape:

$$\otimes_e\frac{\text{pop}\dfrac{}{\vdash \text{pop} : \omega\sigma \multimap \sigma \otimes \omega\sigma} \quad \text{stream}\dfrac{\vdash \mathbf{M}(0) : \sigma \quad \vdash \mathbf{M}(1) : \sigma \quad \ldots \quad \vdash \mathbf{M}(n) : \sigma \quad \ldots}{\vdash \mathbf{M} : \omega\sigma}}{\vdash \text{pop}\,\mathbf{M} : \omega\sigma \otimes \omega\sigma}$$

We set $\mathcal{D}_2$ as the following typing derivation:

$$\otimes\frac{\vdots \quad \text{stream}\dfrac{\vdash \mathbf{M}(1) : \sigma \quad \vdash \mathbf{M}(2) : \sigma \quad \ldots \quad \vdash \mathbf{M}(n+1) : \sigma \quad \ldots}{\vdash tl(\mathbf{M}) : \omega\sigma}}{\vdash \mathbf{M}(0) : \sigma \quad \vdash \mathbf{M}(0) \otimes tl(\mathbf{M}) : \sigma \otimes \omega\sigma}$$

It is easy to check that $\mathcal{D}_1^\dagger \to^*_{\text{cut}} \mathcal{D}_2^\dagger$.

Let $M_1 = (\lambda x.P)N$ and $M_2 = P[N/x]$. By inspecting the typing rules in Figure 13 $\mathcal{D}$ must have the following structure:

$$\frac{\multimap_e\dfrac{\overbrace{\Sigma' \vdash \lambda x.P' : \tau \multimap B'}^{\mathcal{D}} \quad \overbrace{\Delta' \vdash Q' : \tau}^{\mathcal{D}_2}}{\Sigma', \Delta' \vdash (\lambda x.P')Q' : B'}}{\delta \vdots}{\Sigma, \Delta \vdash (\lambda x.P)Q : \sigma}$$

where:

- $\Gamma = \Sigma, \Delta$ and $\sigma = !.^n.!\forall \vec{X}.B$, for some $n \geq 0$, $\vec{X}$, and B.
- δ is a sequence of rules in $\{?w, ?b, f!p, \forall_i, \forall_e\}$,
- $B = B'[\vec{C}/\vec{Y}]$, for some $\vec{C}$ and $\vec{Y}$ not free in Σ', Δ'
- $P'[\vec{x}/\vec{y}] = P$ and $Q'[\vec{x}/\vec{y}] = Q$, for some $\vec{x}, \vec{y}$.

$$\text{ax}\ \frac{}{x : A \vdash x : A} \quad\mapsto\quad \text{ax}\ \frac{}{A^\perp, A} \qquad\qquad \multimap_i \frac{\Gamma, x : \sigma \vdash M : B}{\Gamma \vdash \lambda x.M : \sigma \multimap B} \quad\mapsto\quad \invamp\ \frac{\Gamma^{\dagger\perp}, \sigma^{\dagger\perp}, B^\dagger}{\Gamma^{\dagger\perp}, (\sigma \multimap B)^\dagger}$$

$$\multimap_e \frac{\Gamma \vdash M : \sigma \multimap B \quad \Delta \vdash N : \sigma}{\Gamma, \Delta \vdash MN : B} \quad\mapsto\quad \text{cut}\ \frac{\Gamma^{\dagger\perp}, (\sigma \multimap B)^\dagger \qquad \otimes\dfrac{\Delta^{\dagger\perp}, A^\dagger \quad \text{ax}\ \dfrac{}{B^{\dagger\perp}, B^\dagger}}{\Delta^{\dagger\perp}, A^\dagger \otimes B^{\dagger\perp}, B^\dagger}}{\Gamma^{\dagger\perp}, \Delta^{\dagger\perp}, B^\dagger} \qquad \mathsf{I}_i\ \frac{}{\vdash \mathsf{I} : \mathbf{1}} \quad\mapsto\quad \mathbf{1}\ \frac{}{\mathbf{1}}$$

$$\mathsf{I}_e \frac{\Gamma \vdash N : \mathbf{1} \quad \Delta \vdash M : \sigma}{\Gamma, \Delta \vdash \mathsf{let\ I} = N \mathsf{\ in\ } M : \sigma} \quad\mapsto\quad \text{cut}\ \frac{\Gamma^{\dagger\perp}, \mathbf{1}^\dagger \quad \perp\dfrac{\Delta^{\dagger\perp}, \sigma^\dagger}{\perp, \Delta^{\dagger\perp}, \sigma^\dagger}}{\Gamma^{\dagger\perp}, \Delta^{\dagger\perp}, \sigma^\dagger} \qquad \otimes_i \frac{\Gamma \vdash M : \sigma \quad \Delta \vdash N : \tau}{\Gamma, \Delta \vdash M \otimes N : \sigma \otimes \tau} \quad\mapsto\quad \otimes\ \frac{\Gamma^{\dagger\perp}, \sigma^\dagger \quad \Delta^{\dagger\perp}, \tau^\dagger}{\Gamma^{\dagger\perp}, \Delta^{\dagger\perp}, (\sigma \otimes \tau)^\dagger}$$

$$\otimes_e \frac{\Gamma \vdash M \otimes N : \sigma \otimes \tau \quad \Delta, x : \sigma, y : \tau \vdash P : C}{\Gamma, \Delta \vdash \mathsf{let\ } x \otimes y = M \otimes N \mathsf{\ in\ } P : C} \quad\mapsto\quad \text{cut}\ \frac{\Gamma^{\dagger\perp}, (\sigma \otimes \tau)^\dagger \quad \invamp\dfrac{\Delta^{\dagger\perp}, \sigma^{\dagger\perp}, \tau^{\dagger\perp}, C^\dagger}{\Delta^{\dagger\perp}, \sigma^{\dagger\perp} \invamp \tau^{\dagger\perp}, C^\dagger}}{\Gamma^{\dagger\perp}, \Delta^{\dagger\perp}, C^\dagger} \qquad \forall_i \frac{\Gamma \vdash M : A}{\Gamma \vdash M : \forall X.A} \quad\mapsto\quad \forall\ \frac{\Gamma^{\dagger\perp}, A^\dagger}{\Gamma^{\dagger\perp}, (\forall X.A)^\dagger}$$

$$\forall_e \frac{\Gamma \vdash M : \forall X.A}{\Gamma \vdash M : A[B/X]} \quad\mapsto\quad \text{cut}\ \frac{\Gamma^{\dagger\perp}, (\forall X.A)^\dagger \quad \exists\dfrac{\text{ax}\ \dfrac{}{A^{\dagger\perp}[B^\dagger/X], A^\dagger[B^\dagger/X]}}{\exists X.A^{\dagger\perp}, A^\dagger[B^\dagger/X]}}{\Gamma^{\dagger\perp}, (A[B/X])^\dagger} \qquad \mathsf{f!p}\ \frac{\Gamma \vdash M : \sigma}{!\Gamma \vdash M : !\sigma} \quad\mapsto\quad \mathsf{f!p}\ \frac{\Gamma^{\dagger\perp}, \sigma^\dagger}{!\Gamma^{\dagger\perp}, (!\sigma)^\dagger}$$

$$?\mathsf{w}\ \frac{\Gamma \vdash M : \tau}{\Gamma, x : !\sigma \vdash M : \tau} \quad\mapsto\quad ?\mathsf{w}\ \frac{\Gamma^{\dagger\perp}, \tau^\dagger}{\Gamma^{\dagger\perp}, (!\sigma)^{\dagger\perp}, \tau^\dagger} \qquad ?\mathsf{b}\ \frac{\Gamma, y : \sigma, z : !\sigma \vdash M : \tau}{\Gamma, x : !\sigma \vdash M[x/y, x/z] : \tau} \quad\mapsto\quad ?\mathsf{b}\ \frac{\Gamma^{\dagger\perp}, \sigma^{\dagger\perp}, ?\sigma^{\dagger\perp}, \tau^\dagger}{\Gamma^{\dagger\perp}, (!\sigma)^{\dagger\perp}, \tau^\dagger}$$

Figure 16: Translation from PTA$_2$ to PLL$_2$.

$$\text{stream}\ \frac{\vdash \mathbf{M}(0) : \sigma \quad \vdash \mathbf{M}(1) : \sigma \quad \ldots \quad \vdash \mathbf{M}(n) : \sigma \quad \ldots}{\vdash \mathbf{M} : \omega\sigma} \quad\mapsto\quad \text{ib!p}\ \frac{\sigma^\dagger \quad \sigma^\dagger \quad \ldots \quad \sigma^\dagger \quad \ldots}{(\omega\sigma)^\dagger}$$

$$\text{disc}\ \frac{}{\vdash \mathsf{disc} : \omega\sigma \multimap \mathbf{1}} \quad\mapsto\quad \invamp\frac{?\mathsf{w}\dfrac{\mathbf{1}\dfrac{}{\mathbf{1}^\dagger}}{(\omega\sigma)^{\dagger\perp}, \mathbf{1}^\dagger}}{(\omega\sigma \multimap \mathbf{1})^\dagger} \qquad\qquad \text{pop}\ \frac{}{\vdash \mathsf{pop} : \omega\sigma \multimap \sigma \otimes \omega\sigma} \quad\mapsto\quad \invamp\frac{?\mathsf{b}\dfrac{\otimes\dfrac{\text{ax}\ \dfrac{}{\sigma^{\dagger\perp}, \sigma^\dagger} \quad \mathsf{f!p}\dfrac{\text{ax}\ \dfrac{}{\sigma^{\dagger\perp}, \sigma^\dagger}}{(\omega\sigma)^{\dagger\perp}, !\sigma^\dagger}}{\sigma^{\dagger\perp}, (\omega\sigma)^{\dagger\perp}, \sigma^\dagger \otimes !\sigma^\dagger}}{\sigma^{\dagger\perp}, (\omega\sigma)^{\dagger\perp}, \sigma^\dagger \otimes !\sigma^\dagger}}{(\omega\sigma)^{\dagger\perp}, (\sigma \otimes \omega\sigma)^\dagger}}{(\omega\sigma \multimap \sigma \otimes \omega\sigma)^\dagger}$$

Figure 17: Translation from nuPTA$_2$ to nuPLL$_2$.

By a similar reasoning, $\mathcal{D}_1$ has the following shape:

$$\multimap_i \frac{\overset{\displaystyle \mathcal{D}_1'}{\overline{\Sigma'', x : \tau' \vdash P'' : B''}}}{\dfrac{\dfrac{\Sigma'' \vdash \lambda x.P'' : \tau' \multimap B''}{\varepsilon\ \vdots}}{\Sigma' \vdash \lambda x.P' : \tau \multimap B'}}$$

where:

- ε is sequences of typing rules in $\{?\mathsf{w}, ?\mathsf{b}, \forall_i, \forall_e\}$,
- $B' = B''[\vec{D}/\vec{Z}]$ and $\tau = \tau'[\vec{D}/\vec{Z}]$, for some $\vec{D}$ and $\vec{Z}$ not free in Σ''
- $P''[\vec{z}/\vec{w}] = P$ for some $\vec{z}, \vec{w}$.

Since $\tau = \tau'[\vec{D}/\vec{Z}]$, $B' = B''[\vec{D}/\vec{Z}]$ and $\vec{Z}$ do not occur free in Σ'', by Lemma 71 there is a typing derivation $\mathcal{D}_1''$ of $\Sigma'', x : \tau \vdash P'' : B'$. By Lemma 72 there is a typing derivation $S(\mathcal{D}_1'', \mathcal{D}_2)$ of $\Delta', \Sigma'' \vdash P''[Q'/x] : B'$. Finally, by applying the sequences of rules δ and ε we obtain:

$$\widehat{\mathcal{D}} \ := \ \frac{\overset{\displaystyle S(\mathcal{D}_1'', \mathcal{D}_2)}{\overline{\Delta', \Sigma'' \vdash P''[Q'/x] : B'}}}{\dfrac{\dfrac{\vdots}{\Delta', \Sigma' \vdash P'[Q'/x] : B'}}{\dfrac{\vdots}{\Sigma, \Delta \vdash P[Q/x] : \sigma}}}$$

Let us now show that $\mathcal{D}^\dagger \to^*_{\text{cut}} \widehat{\mathcal{D}}^\dagger$. First, notice that $\mathcal{D}^\dagger$ is as follows:

$$
\cfrac{
\cfrac{
\cfrac{
\cfrac{(\Sigma'')^{\dagger\perp}, (\tau')^{\dagger\perp}, (B'')^\dagger}{(\Sigma'')^{\dagger\perp}, (\tau')^{\dagger\perp} \,\mathfrak{N}\, (B'')^\dagger} \,\raisebox{1ex}{$\overline{(\mathcal{D}'_1)^\dagger}$}
}{
\cfrac{\varepsilon^\dagger \;\vdots}{(\Sigma')^{\dagger\perp}, \tau^{\dagger\perp} \,\mathfrak{N}\, (B')^\dagger}
}
\;\otimes\;
\cfrac{(\Delta')^{\dagger\perp}, \tau^\dagger, \quad \text{ax}\,\overline{(B')^{\dagger\perp}, (B')^\dagger}}{(\Delta')^{\dagger\perp}, \tau^\dagger \otimes (B')^{\dagger\perp}, (B')^\dagger}\,\raisebox{1ex}{$\overline{\mathcal{D}_2^\dagger}$}
}{
\text{cut}\;\cfrac{(\Sigma')^{\dagger\perp}, (\Delta')^{\dagger\perp}, (B')^\dagger}{}
}
}{
\cfrac{\delta^\dagger \;\vdots}{\Sigma^{\dagger\perp}, \Delta^{\dagger\perp}, \sigma^\dagger}
}
$$

Moreover, since $\tau = \tau'[\vec{D}/\vec{Z}]$, $B' = B''[\vec{D}/\vec{Z}]$ and $\vec{Z}$ do not occur free in Σ'', the above derivation reduces by cut-elimination to the following:

$$
\cfrac{
\cfrac{(\Delta')^{\dagger\perp}, \tau^\dagger, \quad (\Sigma'')^{\dagger\perp}, \tau^{\dagger\perp}, (B')^\dagger}{(\Delta')^{\dagger\perp}, (\Sigma'')^{\dagger\perp}, (B')^\dagger}\;\text{cut}\;\;\raisebox{1ex}{$\overline{\mathcal{D}_2^\dagger}\quad\overline{(\mathcal{D}''_1)^\dagger}$}
}{
\cfrac{\vdots}{
\cfrac{(\Delta')^{\dagger\perp}, (\Sigma')^{\dagger\perp}, (B')^\dagger}{
\cfrac{\vdots}{\Delta^{\dagger\perp}, \Sigma^{\dagger\perp}, \sigma^\dagger}
}
}
}
$$

for some $\varepsilon^\dagger$ and $\delta^\dagger$. By Lemma 89 the above derivation reduces by cut-elimination to the following:

$$
\left(
\cfrac{
\cfrac{
\overline{\Delta', \Sigma'' \vdash P''[Q'/x] : B'}\;\raisebox{1ex}{$S(\mathcal{D}''_1, \mathcal{D}_2)$}
}{
\cfrac{\vdots}{(\Delta')^{\dagger\perp}, (\Sigma')^{\dagger\perp}, (B')^\dagger}
}
}{
\cfrac{\vdots}{\Delta^{\dagger\perp}, \Sigma^{\dagger\perp}, \sigma^\dagger}
}
\right)^\dagger
$$

which is $\widehat{\mathcal{D}}^\dagger$. $\qquad\square$

THEOREM 57. *Let $f : (\{0,1\}^*)^n \to \{0,1\}^*$:*

(1) *If f is representable in nuPTA_2 then so it is in nuPLL_2;*
(2) *If f is representable in PTA_2 then it is in PLL_2.*

PROOF. We only consider the case where f is unary for the sake of simplicity. Let $\underline{f}$ be a typable term of nuPTA_2 representing f, so that $\underline{f}\,\underline{s} \to^*_\beta \underline{f(s)}$ for any $s \in \{0,1\}^*$. Consider the following

derivation:

$$
\mathcal{D} = \;\; \cfrac{
\cfrac{\vdash \underline{f} : \text{S}[] \multimap \text{S}[]\;\raisebox{1ex}{$\overline{\mathcal{D}_f}$} \qquad \vdash \underline{s} : \text{S}[]\;\raisebox{1ex}{$\overline{\mathcal{D}_s}$}}{}
}{
\multimap_e\;\cfrac{}{\vdash \underline{f}\,\underline{s} : \text{S}[]}
}
$$

By repeatedly applying Lemma 90 there is $\mathcal{D}_{f(s)}$ such that

$$
\mathcal{D}^\dagger = \;\;
\cfrac{
\raisebox{1ex}{$\overline{\mathcal{D}_f^\dagger}$}\;\;\text{S}[] \multimap \text{S}[] \;\otimes\; \cfrac{\text{S}[]\;\raisebox{1ex}{$\overline{\mathcal{D}_s^\dagger}$}\quad \text{ax}\,\overline{\text{S}[]^\perp, \text{S}[]}}{\text{S}[] \otimes \text{S}[]^\perp, \text{S}[]}
}{
\text{cut}\;\cfrac{}{\text{S}[]}
}
\;\to^*_{\text{cut}}\;\;
\raisebox{0ex}{$\overline{\mathcal{D}_{f(s)}^\dagger}$}\;\;\text{S}[]
$$

in nuPLL_2, where we can safely assume that $\mathcal{D}_s^\dagger \to^*_{\text{cut}} \underline{s}$ and $\mathcal{D}_{f(s)}^\dagger \to^*_{\text{cut}} \underline{f(s)}$ in nuPLL_2. This means that $\mathcal{D}_f^\dagger$ represents f in nuPLL_2. If moreover $\underline{f}$ is typable term of PTA_2 then $\mathcal{D}_f^\dagger$ represents f in PLL_2. $\qquad\square$

D.5 Some remarks on the type system

The following observations justify the finiteness condition on the typing rule ib!p, and the restriction on second-order instantiation to $(!, \omega)$-free types in nuPTA_2.

Remark 91. If the side condition on the typing rule stream (i.e., that $\{\mathbf{M}(i) \mid i \in \mathbb{N}\}$ is finite) were dropped, then nuPTA_2 would represent any function on natural numbers. Indeed, given a function $f : \mathbb{N} \to \mathbb{N}$, we can define the term $\mathbf{F} := \underline{f(0)} :: \underline{f(1)} :: \ldots$ with type $\omega!\mathbf{N}$, encoding all values of the function f. We set $A := \mathbf{N}[1] \otimes \omega\mathbf{N}[1]$ and define:

$$
\text{step} := \lambda x.\text{let } y_1 \otimes y_2 = x \text{ in let } \mathsf{I} = y_1\,(\lambda z.z)\,\mathsf{I} \text{ in pop } y_2
$$
$$
\underline{f} := \lambda n.\text{let } x \otimes y = (n\,\text{step}\,(\text{pop } \mathbf{F})) \text{ in let } \mathsf{I} = (\text{disc } y) \text{ in } x
$$

where step has type $A \multimap A$ and $\underline{f}$ has type $\mathbf{N}[A] \multimap \mathbf{N}[1]$. It is easy to check that $\underline{f}\,\underline{n} \to^*_\beta \underline{f(n)}$, for any $n \in \mathbb{N}$.

This observation can be easily adapted to the proof systems nuPLL_2 (w.r.t. the finiteness condition on ib!p) and wrPLL_2^∞ (w.r.t. the weak regularity condition).

Remark 92. If the $(!, \omega)$-freeness condition on $\forall_e$ were dropped then nuPTA_2 could represent exponential functions. Indeed, we can define the following functions:

$$
\begin{aligned}
\text{plustwo} &:= \lambda n.\lambda f.\lambda z.nf(f(fz)) &: \mathbf{N}[] \multimap \mathbf{N}[] \\
\text{double} &:= \lambda n.n\,(\text{plustwo})\,\underline{0} &: \mathbf{N}[\mathbf{N}[]] \multimap \mathbf{N}[] \\
\text{exp} &:= \lambda n.n\,(\text{double})\,\underline{1} &: \mathbf{N}[\mathbf{N}[]] \multimap \mathbf{N}[]
\end{aligned}
$$

It is easy to check that, for any $n \in \mathbb{N}$:

$$
\text{plustwo}\,\underline{n} \to^*_\beta \underline{n+2} \qquad \text{double}\,\underline{n} \to^*_\beta \underline{2n} \qquad \text{exp}\,\underline{n} \to^*_\beta \underline{2^n}
$$

E RELATIONAL SEMANTICS FOR NON-WELLFOUNDED PROOFS

Here we define a denotational model for oPLL_2^∞ based on the *relational semantics*, which interprets an open coderivation as the union of the interpretations of its finite approximations, as in [25].

$$\left[\!\!\left[\, \mathrm{hyp} \, \frac{}{\Gamma} \right]\!\!\right]_n = \varnothing \qquad \left[\!\!\left[\, \mathrm{ax} \, \frac{}{A, A^\perp} \right]\!\!\right]_n = \left\{\, (x, x) \mid x \in [\![A]\!] \,\right\} \qquad \left[\!\!\left[\, \mathrm{cut} \, \frac{\overset{\mathcal{D}'}{\nabla}\Gamma, A \quad \overset{\mathcal{D}''}{\nabla}\Delta, A^\perp}{\Gamma, \Delta} \right]\!\!\right]_n = \left\{\, (\vec{x}, \vec{y}) \;\middle|\; \exists z \in [\![A]\!] \text{ s.t. } \begin{array}{c} (\vec{x}, z) \in [\![\mathcal{D}']\!]_{n-1} \\ \text{and} \\ (z, \vec{y}) \in [\![\mathcal{D}'']\!]_{n-1} \end{array} \right\}$$

$$\left[\!\!\left[\, \otimes \, \frac{\overset{\mathcal{D}'}{\nabla}\Gamma, A \quad \overset{\mathcal{D}''}{\nabla}\Delta, B}{\Gamma, \Delta, A \otimes B} \right]\!\!\right]_n = \left\{\, (\vec{x}, \vec{y}, (x, y)) \;\middle|\; \begin{array}{c} (\vec{x}, x) \in [\![\mathcal{D}']\!]_{n-1} \\ \text{and} \\ (\vec{y}, y) \in [\![\mathcal{D}'']\!]_{n-1} \end{array} \right\} \qquad \left[\!\!\left[\, \mathfrak{N} \, \frac{\overset{\mathcal{D}'}{\nabla}\Gamma, A, B}{\Gamma, A \,\mathfrak{N}\, B} \right]\!\!\right]_n = \left\{\, (\vec{x}, (y, z)) \mid (\vec{x}, y, z) \in [\![\mathcal{D}']\!]_{n-1} \,\right\}$$

$$\left[\!\!\left[\, 1 \, \frac{}{1} \right]\!\!\right]_n = \{*\} \qquad \left[\!\!\left[\, \perp \, \frac{\overset{\mathcal{D}'}{\nabla}\Gamma}{\Gamma, \perp} \right]\!\!\right]_n = \left\{\, (\vec{x}, *) \mid \vec{x} \in [\![\mathcal{D}']\!]_{n-1} \,\right\} \qquad \left[\!\!\left[\, \forall \, \frac{\overset{\mathcal{D}'}{\nabla}\Gamma, A}{\Gamma, \forall X.A} \right]\!\!\right]_n = \left[\!\!\left[\, \exists \, \frac{\overset{\mathcal{D}'}{\nabla}\Gamma, A[B/X]}{\Gamma, \exists X.A} \right]\!\!\right]_n = [\![\mathcal{D}']\!]_{n-1}$$

$$\left[\!\!\left[\, ?\mathrm{w} \, \frac{\overset{\mathcal{D}'}{\nabla}\Gamma}{\Gamma, ?A} \right]\!\!\right]_n = \left\{\, (\vec{x}, [\,]) \mid \vec{x} \in [\![\mathcal{D}']\!]_{n-1} \,\right\} \qquad \left[\!\!\left[\, ?\mathrm{b} \, \frac{\overset{\mathcal{D}'}{\nabla}\Gamma, A, ?A}{\Gamma, ?A} \right]\!\!\right]_n = \left\{\, (\vec{x}, [y] + \mu) \mid (\vec{x}, y, \mu) \in [\![\mathcal{D}']\!]_{n-1} \,\right\}$$

$$\left[\!\!\left[\, \mathrm{c!p} \, \frac{\overset{\mathcal{D}'}{\nabla}\Gamma, A \quad \overset{\mathcal{D}''}{\nabla}?\Gamma, !A}{?\Gamma, !A} \right]\!\!\right]_n = \left\{(\vec{[\,]}, [\,])\right\} \cup \left\{\, ([x_1] + \mu_1, \ldots, [x_k] + \mu_k, [x] + \mu) \;\middle|\; \begin{array}{c} (x_1, \ldots, x_k, x) \in [\![\mathcal{D}']\!]_{n-1} \\ \text{and} \\ (\mu_1, \ldots, \mu_k, \mu) \in [\![\mathcal{D}'']\!]_{n-1} \end{array} \right\}$$

Figure 18: Inductive definition of the set $[\![\mathcal{D}]\!]_n$, for $n > 0$.

The relational semantics interprets the exponentials by finite multisets, denoted by brackets, e.g., $[x_1, \ldots, x_n]$; $+$ denotes the *multiset union*, $\mathcal{M}_f(X)$ denotes the set of finite multisets over a set X. To correctly define the semantics of a coderivation, we need to see sequents as *finite sequence* of formulas (taking their order into account), which means that we have to add an *exchange* rule to oPLL_2^∞ to swap the order of two consecutive formulas in a sequent.

Definition 93 (Reflexive object)**.** We define $D := \bigcup_{n \in \mathbb{N}} D_n$, where D_n is defined by induction as follows:

$$\begin{aligned} D_0 &:= \{*\} \\ D_{n+1} &:= D_0 \cup (D_n \times D_n) \cup \mathcal{M}_f(D_n) \end{aligned}$$

Definition 94. We associate with each formula A a **set** $[\![A]\!]$ defined as follows:

$$\begin{aligned} [\![X]\!] &:= D & [\![A \otimes B]\!] &:= [\![A]\!] \times [\![B]\!] \\ [\![1]\!] &:= \{*\} & [\![!A]\!] &:= \mathcal{M}_f([\![A]\!]) \\ [\![A^\perp]\!] &:= [\![A]\!] & [\![\forall X.A]\!] &:= [\![A]\!] \end{aligned}$$

where D is as in Definition 93. For a sequent $\Gamma = A_1, \ldots, A_n$, we set $[\![\Gamma]\!] := [\![A_1 \,\mathfrak{N} \cdots \mathfrak{N}\, A_n]\!]$.

Given oPLL_2^∞ with conclusion Γ, we set $[\![\mathcal{D}]\!] := \bigcup_{n \geq 0} [\![\mathcal{D}]\!]_n \subseteq [\![\Gamma]\!]$, where $[\![\mathcal{D}]\!]_0 = \varnothing$ and, for all $i \in \mathbb{N} \setminus \{0\}$, $[\![\mathcal{D}]\!]_i$ is defined inductively according to Figure 18.

Example 95. For the coderivations $\mathcal{D}_\perp$ and $\mathcal{D}_?$ in Figure 3, we have $[\![\mathcal{D}_\perp]\!] = [\![\mathcal{D}_?]\!] = \varnothing$. For the derivations $\underline{0}$ and $\underline{1}$ in Figure 3, $[\![\underline{0}]\!] = \{((x, y), (x, y)) \mid x \in D\}$ and $[\![\underline{1}]\!] = \{((x, y), (y, x)) \mid x, y \in D\}$. For the coderivation $\mathrm{c!p}_{(\underline{i_0}, \ldots, \underline{i_n}, \ldots)}$ in Example 14 (with $i_j \in \{0, 1\}$ for

all $j \in \mathbb{N}$), the relation $\left[\!\!\left[\mathrm{c!p}_{(\underline{i_0}, \ldots, \underline{i_n}, \ldots)}\right]\!\!\right]$ is defined as the set of all multisets of the form $[((x_1, y_1), (z_1, w_1)), \ldots, ((x_n, y_n), (z_n, w_n))]$ with $n \in \mathbb{N}$ and $x_j, y_j, z_j, w_j \in D$ such that $x_j = w_j$ and $y_j = z_j$ whenever $\underline{i_j} = \underline{1}$, and such that $x_j = z_j$ and $y_j = w_j$ whenever $\underline{i_j} = \underline{0}$.

By straightforward inspection of the cut-elimination steps for oPLL_2^∞ we have:

Theorem 96 (Soundness)**.** *Let $\mathcal{D} \in \mathrm{oPLL}_2^\infty$. If $\mathcal{D} \to_{\mathrm{cut}} \mathcal{D}'$, then $[\![\mathcal{D}]\!] = [\![\mathcal{D}']\!]$.*