NON-UNIFORM POLYNOMIAL TIME AND NON-WELLFOUNDED PARSIMONIOUS PROOFS

MATTEO ACCLAVIO[1], GIANLUCA CURZI[2], AND GIULIO GUERRIERI[3]

ABSTRACT. In this paper we investigate the complexity-theoretical aspects of non-wellfounded proof systems inspired by parsimonious logic, a variant of linear logic where the exponential modality ! is interpreted as a constructor for streams over finite data. Specifically, we introduce the non-wellfounded proof systems rPLL_2^∞ and wrPLL_2^∞, where logical consistency is maintained at a global level by adapting a standard progressing criterion, and we provide a denotational semantics based on the relational model. We prove that rPLL_2^∞ and wrPLL_2^∞ characterise, respectively, the complexity classes **FP** and **FP**/poly. As a byproduct of our proof methods, we establish a series of characterisation results for finitary proof systems.

1. INTRODUCTION

Non-wellfounded proof theory studies proofs as possibly infinite (but finitely branching) trees, where logical consistency is maintained by means of global conditions called *progressing* (or *validity*) *criteria*. In this setting, the so-called *regular* (also called *circular* or *cyclic*) proofs receive a special attention, in that they admit a finite description based on (possibly cyclic) directed graphs.

This area of proof theory makes its first appearance (in its modern guise) in the context of the modal μ-calculus [1, 2]. Since then, it has been extensively investigated from many perspectives, such as predicate logic [3, 4], algebras [5, 6], arithmetic [7, 8, 9], proofs-as-programs interpretations [10, 11, 12, 13, 14], and continuous cut-elimination [15, 16], establishing itself as an ideal setting for manipulating least and greatest fixed points, and hence for modelling induction and coinduction principles. In particular, in [12, 13, 14] non-wellfounded proof-theory has been investigated under 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 plays the role of a totality requirement (i.e., functions computed by progressing proofs are always well-defined on all arguments). On the other hand, the regularity condition 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.

[1]DEPARTMENT OF COMPUTER SCIENCE, UNIVERSITY OF SOUTHERN DENMARK, MACCLAVIO@GMAIL.COM

[2]SCHOOL OF COMPUTER SCIENCE, UNIVERSITY OF BIRMINGHAM, BIRMINGHAM, UK, G.CURZI@BHAM.AC.UK

[3]LIS, AIX MARSEILLE UNIVERSITÉ, GIULIO.GUERRIERI@LIS-LAB.FR

In joint work with Das [17], the second author extended this computational reading of non-wellfounded proofs to the realm of *computational complexity*, by introducing circular proof systems capturing, respectively, the class of functions computable in polynomial time (**FP**) and the elementary functions (**FELEMENTARY**). These proof systems are defined by identifying global conditions on circular progressing proofs motivated by ideas from *Implicit Computational Complexity* (ICC), 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 [18], one of the cornerstones of ICC.

In a follow up paper [19], the second author and Das generalised the characterisation results in [17] to capture $\mathbf{FP}/\mathsf{poly}$, i.e., the class of functions computable in polynomial time by Turing machines with access to *polynomial advice* or, equivalently, computable by non-uniform families of polynomial-size circuits [20]. Specifically, *non-uniform complexity* is modelled by more permissive non-wellfounded proof systems (compared to circular proof systems), essentially obtained by weakening the regularity condition, hence relaxing finite presentability of proofs. Note, indeed, that $\mathbf{FP}/\mathsf{poly}$ includes *undecidable problems*, and so cannot be characterised by purely circular proof systems, which typically represent only computable functions.

In this paper we rather follow an alternative route to $\mathbf{FP}/\mathsf{poly}$ based on *linear logic* [21]. 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 those 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 (but finite) number of times an input of type A.

The approaches to ICC based on linear logic have spurred a variety of methods for taming complexity. The central idea is to weaken the exponential rules in order to induce a bound on cut-elimination, hence reducing the computational strength of the system. These weaker versions of linear logic, often called *light logics*, are typically formulated in a second-order framework, using type polymorphism for iterating resource-bounded Turing machine transition functions, a crucial step for proving completeness w.r.t. a complexity class. Over the years, several light logics have been proposed to characterise major complexity classes. Examples are *soft linear logic* (SLL) or *light linear logic* (LLL) [22, 23] for **FP**, and *elementary linear logic* (ELL) [24, 25] for **FELEMENTARY**.

Continuing this tradition, in a series of papers [26, 27] 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., $!A \multimapboth A \otimes !A$) and invalidates the implications $!A \multimap !!A$ (*digging*) and $!A \multimap !A \otimes !A$ (*contraction*). In parsimonious logic, a proofs of $!A$ can be interpreted as a *stream* over (a finite set of) proofs of A, i.e., as a greatest fixed point, where the linear implications $A \otimes !A \multimap !A$ (*co-absorption*) and $!A \multimap A \otimes !A$ (*absorption*) can be computationally read as the *push* and *pop* operations on streams.

In joint work with Terui [27], Mazza introduced a second-order non-uniform version of PL, and showed that the resulting system, called $\mathsf{nuPL}_{\forall\ell}$, captures the problems decidable by non-uniform families of circuits ($\mathbf{P}/\mathsf{poly}$). Specifically, $\mathsf{nuPL}_{\forall\ell}$ models non-uniformity via an *infinitely branching rule* that takes a finite set of proofs $\mathcal{D}_1, \ldots, \mathcal{D}_n$ of A and a (possibly non-recursive) function $f : \mathbb{N} \to \{1, \ldots, n\}$ as premises, and constructs a proof of $!A$ representing a stream of proofs of the form $\mathfrak{S} = (\mathcal{D}_{f(0)}, \mathcal{D}_{f(1)}, \ldots, \mathcal{D}_{f(n)}, \ldots)$. On the one hand, thanks to this rule, $\mathsf{nuPL}_{\forall\ell}$ can express Turing machines with *advices*. On the other hand, polynomial step cut-elimination is guaranteed thanks to the absence of digging and contraction principles. As a byproduct of their result, second-order (uniform) parsimonious logic, i.e. $\mathsf{PL}_{\forall\ell}$, characterises the class of problems decidable in polynomial time ($\mathbf{P}$)[1]. Hence, parsimonious logic exponential modalities exploit in an essential way the above-mentioned proof-theoretical non-uniformity, which in turn deeply interfaces with notions of non-uniformity from computational complexity [27].

The analysis of parsimonious logic conducted in [26, 27] reveals that fixed point definitions of the exponentials are better behaving when digging and contraction are discarded. On the other hand, the co-absorption rule cannot be derived in LL, and so it prevents parsimonious logic becoming a genuine subsystem of the latter. This led the authors of the present paper to introduce *parsimonious linear logic*, a co-absorption-free subsystem of linear logic that nonetheless allows a stream-based interpretation of the exponentials.

In this paper we investigate a non-wellfounded version of second-order parsimonious logic, where "streams" of proofs such as $\mathfrak{S}$ defined by the infinitely branching rule $!\mathsf{I}$ are modelled by (finitely branching but) non-wellfounded proofs. The resulting system is *non-wellfounded parsimonious linear logic* (PLL_2^∞). To avoid fallacious reasoning and to recover logical consistency we adapt to our setting the *progressing criterion*, which relies on threads of exponential formulas occurring in the infinite branches of a prooftree. The restriction of PLL_2^∞ to progressing proofs, called pPLL_2^∞, allows an interpretation of the modal formulas $!A$ and its dual $?A^\perp$ in terms of greatest and least fixed points respectively[2].

As it stands, pPLL_2^∞ far exceeds the computational strength of Mazza's systems, as nothing guarantees that non-wellfounded proofs of $!A$ represent streams over *finite* sets of data of type A, a built-in feature of the rule $!\mathsf{I}$. This is achieved by imposing further global requirements on the structure of non-wellfounded proofs, obtaining the systems *non-uniform parsimonious linear logic* (wrPLL_2^∞) and *circular parsimonious linear logic* (rPLL_2^∞). Specifically, in analogy with [19], rPLL_2^∞ is a system of circular proofs, while wrPLL_2^∞ is defined via a *weak regularity* condition. Roughly, wrPLL_2^∞ and rPLL_2^∞ correspond to the systems $\mathsf{nuPL}_{\forall\ell}$ and $\mathsf{PL}_{\forall\ell}$ respectively. Thus, the distinction between regularity and weak regularity duly reflects the interplay between uniform and non-uniform computation.

[1]Despite not being explicitly stated in [27], the latter statement is a straightforward consequence of the characterisation result for $\mathsf{nuPL}_{\forall\ell}$.

[2]Note that definitions of exponentials based on fixed points have been proposed in the context of $\mu\mathsf{MALL}$ by defining $!A \coloneqq \nu X.(\mathbf{1} \,\&\, A \,\&\, (X \otimes X))$ and $?A \coloneqq \mu X.(\perp \oplus A \oplus (X \parr X))$, where ν and μ are the greatest and the least fixed point operator respectively [28]. However, as shown in [29], such an encoding does not give rise to a Seely category, which is essential to model linear logic. Also, as observed in [28], cut-reductions for the fixed point exponentials and for the standard exponential radically diverge.

On a technical side, wrPLL_2^∞ and rPLL_2^∞ are free of the co-absorption rule from parsimonious logic, so the "push" operation on streams of proofs cannot be modelled in our systems. This fact has a twofold advantage: on the one hand, we improve the complexity-theoretic results of [27], showing that the push constructor is not essential for proving the characterisation theorems; on the other hand, the absence of a co-absorption rule makes the fully-fledged inductive counterparts of wrPLL_2^∞ and rPLL_2^∞, that we call nuPLL_2 and PLL_2, de facto subsystems of linear logic. In fact, to stress this relevant departure from Mazza's parsimonious logic, we use the terminology *parsimonious linear logic* when referring to our (non-)wellfounded systems.

At a complexity-theoretic level, our results contribute to Mazza's work on parsimonious logic in many respects. First, our characterisation results extend those established in [26, 27], as the latter only relate (type) systems with classes of predicated (i.e., $\mathbf{P}/\mathsf{poly}$ and $\mathbf{P}$). Secondly, our non-wellfounded approach to $\mathsf{nuPL}_{\forall\ell}$ has the advantage of trading the constant growth-rate function $f : \mathbb{N} \to \{1, \dots, n\}$, defining the rule $!\mathsf{I}$, for the weak regularity condition, making the characterisation results more "implicit", thus closer to ICC.

Contributions. In this paper we present a series of complexity-theoretic results showing, in particular, that wrPLL_2^∞ and nuPLL_2 duly characterise $\mathbf{FP}/\mathsf{poly}$, while rPLL_2^∞ and PLL_2 characterise $\mathbf{FP}$ (Theorem 46). Out proof methods are summarised by the diagram in Figure 1 relating various (non)wellfounded proof (and type) systems with the complexity classes $\mathbf{FP}/\mathsf{poly}$ and $\mathbf{FP}$. The central result of the diagram is a polynomial *modulus of continuity* for cut-elimination (Lemma 55), from which we infer that wrPLL_2^∞ is sound for $\mathbf{FP}/\mathsf{poly}$, and that rPLL_2^∞ is sound for $\mathbf{FP}$ (Theorem 56). Completeness requires a series of intermediate steps. We first introduce in Section 6.2 the type system nuPTA_2, in which computation can access entries of a stream, and its subsystem PTA_2. Then we show an encoding of polynomial time Turing machines with (polynomial) advice in nuPTA_2, essentially by adapting standard methods from [30, 31] to the setting of non-uniform computation. This allows us to prove that nuPTA_2 is complete for $\mathbf{FP}/\mathsf{poly}$ and that PTA_2 is complete for $\mathbf{FP}$ (Theorem 75). Thirdly, we define computationally sound translations from nuPTA_2 to nuPLL_2, and from PTA_2 to PLL_2 (Theorem 81). Finally, we define computationally sound translations from nuPLL_2 to wrPLL_2^∞, and from PLL_2 to rPLL_2^∞ (Theorem 36). As a byproduct of our series of theorems, we obtain that nuPTA_2 characterises $\mathbf{FP}/\mathsf{poly}$, while PTA_2 characterises $\mathbf{FP}$.

Outline of the paper. This paper is structured as follows. In Section 2 we recall some preliminaries on linear logic, non-wellfounded proofs and non-uniform complexity. In Section 3 we introduce parsimonious linear logic and define the wellfounded proof systems PLL_2, nuPLL_2, and the non-wellfounded proof systems PLL_2^∞, pPLL_2^∞, wrPLL_2^∞, rPLL_2^∞. In Section 5 we define a relational semantics for our proof systems and establish some basic properties of the model. In Section 6 we prove the complexity-theoretic results. In particular, Section 6.1 is devoted to proving the soundness theorem for wrPLL_2^∞ and rPLL_2^∞ (Theorem 56), while Section 6.2 shows the completeness theorem (Theorem 75).

$$\begin{tikzcd}
 & \mathsf{rPLL}_2^\infty \arrow[r, "\subseteq"] \arrow[dl, bend right, "\text{Theorem 56.1}"'] & \mathsf{wrPLL}_2^\infty \arrow[dr, bend left, "\text{Theorem 56.2}"] & \\
\mathbf{FP} \arrow[dr, bend right, "\text{Theorem 75.1}"'] & \mathsf{PLL}_2 \arrow[u, "\text{Theorem 36.1}"] \arrow[r, "\subseteq"] & \mathsf{nuPLL}_2 \arrow[u, "\text{Theorem 36.2}"] & \mathbf{FP}/\mathsf{poly} \arrow[dl, bend left, "\text{Theorem 75.2}"] \\
 & \mathsf{PTA}_2 \arrow[u, "\text{Theorem 81.2}"] \arrow[r, "\subseteq"] & \mathsf{nuPTA}_2 \arrow[u, "\text{Theorem 81.1}"] &
\end{tikzcd}$$

Figure 1. Grand tour diagram.

2. Preliminary notions

In this section we recall some basic notions from (non-wellfounded) proof theory and computational complexity.

2.1. **Derivations and coderivations.** We assume that the reader is familiar with the syntax of sequent calculus, e.g. [32]. Here we specify some conventions adopted to simplify the content of this paper.

In this work we consider (**sequent**) **rules** of the form $\mathsf{r}\frac{}{\Gamma}$ or $\mathsf{r}\frac{\Gamma_1}{\Gamma}$ or $\mathsf{r}\frac{\Gamma_1 \quad \Gamma_2}{\Gamma}$, 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 [33] 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 $v1 \in \mathcal{T}$). Elements of a tree $\mathcal{T}$ are called **nodes** and a node $vn \in \mathcal{T}$ with $n \in \{1,2\}$ is a **child** of $v \in \mathcal{T}$. Given a tree $\mathcal{T}$ and a node $v \in \mathcal{T}$, a **branch** $\mathcal{B}$ of $\mathcal{T}$ (from v) is a set of nodes in $\mathcal{T}$ of the form vw (for any $w \in \{1,2\}^*$) such that if they have at least one child in $\mathcal{T}$ then they have exactly one child in $\mathcal{B}$.

A **coderivation** over a set of rules $\mathcal{S}$ is a labeling $\mathcal{D}$ of a tree by sequents such that if v is a node with children $v_1, \ldots, v_n$ (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 $\mathcal{D}(v_1), \ldots, \mathcal{D}(v_n)$. The **height** of r in $\mathcal{D}$ is the length of the node $v \in \{1,2\}^*$ 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 $\mathcal{D}_v(w) = \mathcal{D}(vw)$.

A coderivation $\mathcal{D}$ is **r-free** (for a rule $\mathsf{r} \in \mathcal{S}$) if it contains no occurrence of r. 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).

Given a set of coderivations X, a sequent Γ is **provable** in X (noted $\vdash_{\mathsf{X}} \Gamma$) if there is a coderivation in X with conclusion Γ.

While derivations are usually represented as finite trees, regular coderivations can be represented as *finite* directed (possibly cyclic) graphs: a cycle is created by linking the roots of two identical subcoderivations.

Definition 2 (Bar)**.** Let $\mathcal{D}$ be a coderivation. A set $\mathcal{V}$ of nodes in $\mathcal{D}$ is a **bar** (of $\mathcal{D}$) if:

- any branch of $\mathcal{D}$ contains a node in $\mathcal{V}$;
- any pair of nodes in $\mathcal{V}$ are mutually incomparable (w.r.t. the partial order in $\mathcal{D}$).

We say that a bar $\mathcal{V}$ has **height** h if every node in $\mathcal{V}$ that is not a leaf of $\mathcal{D}$ has height $\geq h$.

2.2. **Non-uniform complexity classes.** The goal of this paper is a proof theoretic characterisation of the complexity class $\mathbf{FP}/\mathsf{poly}$ [20], that is, 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). Formally, if $\mathbf{FP}$ is class of functions computable in polynomial time by a Turing machine, $\mathbf{FP}/\mathsf{poly}$ is defined as follows.

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

$\mathbf{FP}/\mathsf{poly}$ extends $\mathbf{FP}$ and contains some uncomputable functions, for instance the characteristic function of undecidable unary languages [20, Example 6.4]. The class $\mathbf{FP}/\mathsf{poly}$ can be also defined in terms of non-uniform families of circuits.

Theorem 4 ([20], Thm. 6.11)**.** *A function f is in* $\mathbf{FP}/\mathsf{poly}$ *iff there is polynomial-size familiy of circuits computing f.*

2.3. **An equivalent definition of $\mathbf{FP}/\mathsf{poly}$.** We adopt a different presentation of $\mathbf{FP}/\mathsf{poly}$ that facilitates the proof of completeness.

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

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

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

Proposition 6. *[See, e.g.,* [19]*]* $\mathbf{FP}/\mathsf{poly} = \mathbf{FP}(\mathbb{R})$.

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

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

$$\mathsf{ax}\,\dfrac{}{A, A^\perp} \qquad \mathsf{cut}\,\dfrac{\Gamma, A \quad A^\perp, \Delta}{\Gamma, \Delta} \qquad \parr\,\dfrac{\Gamma, A, B}{\Gamma, A \parr B} \qquad \otimes\,\dfrac{\Gamma, A \quad B, \Delta}{\Gamma, \Delta, A \otimes B} \qquad \mathsf{1}\,\dfrac{}{\mathsf{1}} \qquad \perp\,\dfrac{\Gamma}{\Gamma, \perp}$$

$$\mathsf{f!p}\,\dfrac{\Gamma, A}{?\Gamma, !A} \qquad \mathsf{?w}\,\dfrac{\Gamma}{\Gamma, ?A} \qquad \mathsf{?b}\,\dfrac{\Gamma, A, ?A}{\Gamma, ?A} \qquad \forall\,\dfrac{\Gamma, A}{\Gamma, \forall X.A}\, X \notin FV(\Gamma) \qquad \exists\,\dfrac{\Gamma, A[B/X]}{\Gamma, \exists X.A}\, B \text{ is (!,?)-free}$$

FIGURE 2. Sequent calculus rules of PLL_2.

$\underline{\mathbf{1}}$	$\underline{\mathbf{0}}$	$\mathcal{D}_{\mathsf{abs}}$	$\mathcal{D}_{\mathsf{der}}$
$\forall\,\dfrac{\parr\,\dfrac{\parr\,\dfrac{\otimes\,\dfrac{\mathsf{ax}\,\dfrac{}{X_1^\perp, X_3} \quad \mathsf{ax}\,\dfrac{}{X_2^\perp, X_4}}{X_1^\perp, X_2^\perp, X_3 \otimes X_4}}{(X_1^\perp \parr X_2^\perp), (X_3 \otimes X_4)}}{(X_1^\perp \parr X_2^\perp) \parr (X_3 \otimes X_4)}}{\forall X.(X^\perp \parr X^\perp) \parr (X \otimes X)}$	$\forall\,\dfrac{\parr\,\dfrac{\parr\,\dfrac{\otimes\,\dfrac{\mathsf{ax}\,\dfrac{}{X_1^\perp, X_4} \quad \mathsf{ax}\,\dfrac{}{X_2^\perp, X_3}}{X_1^\perp, X_2^\perp, X_3 \otimes X_4}}{(X_1^\perp \parr X_2^\perp), (X_3 \otimes X_4)}}{(X_1^\perp \parr X_2^\perp) \parr (X_3 \otimes X_4)}}{\forall X.(X^\perp \parr X^\perp) \parr (X \otimes X)}$	$\parr\,\dfrac{\mathsf{?b}\,\dfrac{\otimes\,\dfrac{\mathsf{ax}\,\dfrac{}{A^\perp, A} \quad \mathsf{ax}\,\dfrac{}{?A^\perp, !A}}{A^\perp, ?A^\perp, A \otimes !A}}{?A^\perp, A \otimes !A}}{?A^\perp \parr (A \otimes !A)}$	$\parr\,\dfrac{\mathsf{?b}\,\dfrac{\mathsf{?w}\,\dfrac{\mathsf{ax}\,\dfrac{}{A^\perp, A}}{A^\perp, ?A^\perp, A}}{?A^\perp, A}}{?A^\perp \parr A}$

FIGURE 3. Examples of derivations in PLL_2.

3. SECOND-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:

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

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 (and capture-avoiding) substitution of B for the free occurrences of the propositional variables X in A. **Linear negation** $(\cdot)^\perp$ is defined by De Morgan's laws $(A^\perp)^\perp = A$, $(A \otimes B)^\perp = A^\perp \parr B^\perp$, $(!A)^\perp = ?A^\perp$, $(\mathsf{1})^\perp = \perp$, and $(\forall X.A)^\perp = \exists X.A^\perp$, while **linear implication** is defined as $A \multimap B \coloneqq A^\perp \parr B$.

Definition 7. **Second-order parsimonious linear logic**, denoted by PLL_2, is the set of rules in Figure 2, that is, **axiom** (ax), **cut** (cut), **tensor** ($\otimes$), **par** ($\parr$), **one** ($\mathsf{1}$), **bottom** ($\perp$), **functorial promotion** ($\mathsf{f!p}$), **weakening** ($\mathsf{?w}$), **absorption** ($\mathsf{?b}$), **(second-order) universal quantifier** ($\forall$), **(second-order) existential quantifier** ($\exists$). Rules ax, $\otimes$, $\parr$, $\mathsf{1}$ and $\perp$ are called **multiplicative**, rules $\mathsf{f!p}$, $\mathsf{?w}$ and $\mathsf{?b}$ are called **exponential**. Finally, rules $\forall$ and $\exists$ are called **second-order**. We also denote by PLL_2 the set of derivations over the rules in PLL_2.

Example 8. Figure 3 gives some examples of derivation in PLL_2. The (distinct) derivations $\underline{\mathbf{0}}$ and $\underline{\mathbf{1}}$ prove the same formula $\mathbf{B} = \forall X.(X^\perp \parr X^\perp) \parr (X \otimes X)$, where X_1, X_2, X_3, X_4 are occurrences of the propositional variable X. The derivation $\mathcal{D}_{\mathsf{abs}}$ proves the *absorption law* $!A \multimap A \otimes !A$; the derivation $\mathcal{D}_{\mathsf{der}}$ proves the *dereliction law* $!A \multimap A$.

The **cut-elimination** relation $\to_{\mathsf{cut}}$ in PLL_2 is the union of **principal** cut-elimination steps in Figure 4 (**multiplicative**), Figure 5 (**exponential**) and Figure 6 (**second-order**), and **commutative** cut-elimination steps in Figure 7. The reflexive-transitive closure of $\to_{\mathsf{cut}}$ is noted $\to_{\mathsf{cut}}^*$.

$$\mathsf{cut}\,\dfrac{\mathsf{ax}\,\dfrac{}{A,A^{\perp}} \quad \Gamma,A}{\Gamma,A} \to_{\mathsf{cut}} \Gamma,A \qquad \mathsf{cut}\,\dfrac{\invamp\,\dfrac{\Gamma,A,B}{\Gamma,A\invamp B} \quad \otimes\,\dfrac{\Delta,A^{\perp} \quad B^{\perp},\Sigma}{\Delta,A^{\perp}\otimes B^{\perp},\Sigma}}{\Gamma,\Delta,\Sigma} \to_{\mathsf{cut}} \mathsf{cut}\,\dfrac{\mathsf{cut}\,\dfrac{\Gamma,B,A \quad A^{\perp},\Delta}{\Gamma,\Delta,B} \quad B^{\perp},\Sigma}{\Gamma,\Delta,\Sigma} \qquad \mathsf{cut}\,\dfrac{\perp\,\dfrac{\Gamma}{\Gamma,\perp} \quad \mathsf{1}\,\dfrac{}{1}}{\Gamma} \to_{\mathsf{cut}} \Gamma$$

FIGURE 4. Multiplicative cut-elimination steps in PLL_2.

$$\mathsf{cut}\,\dfrac{\mathsf{f!p}\,\dfrac{\Gamma,A}{?\Gamma,!A} \quad \mathsf{f!p}\,\dfrac{A^{\perp},\Delta,B}{?A^{\perp},?\Delta,!B}}{?\Gamma,?\Delta,!B} \to_{\mathsf{cut}} \mathsf{f!p}\,\dfrac{\mathsf{cut}\,\dfrac{\Gamma,A \quad A^{\perp},\Delta,B}{\Gamma,\Delta,B}}{?\Gamma,?\Delta,!B} \qquad \mathsf{cut}\,\dfrac{\mathsf{f!p}\,\dfrac{\Gamma,A}{?\Gamma,!A} \quad \mathsf{?w}\,\dfrac{\Delta}{\Delta,?A^{\perp}}}{?\Gamma,\Delta} \to_{\mathsf{cut}} \mathsf{?w}\,\dfrac{\Delta}{\overline{?\Gamma,\Delta}}$$

$$\mathsf{cut}\,\dfrac{\mathsf{f!p}\,\dfrac{\Gamma,A}{?\Gamma,!A} \quad \mathsf{?b}\,\dfrac{\Delta,A^{\perp},?A^{\perp}}{\Delta,?A^{\perp}}}{?\Gamma,\Delta} \to_{\mathsf{cut}} \mathsf{?b}\,\dfrac{\mathsf{cut}\,\dfrac{\Gamma,A \quad \mathsf{cut}\,\dfrac{\mathsf{f!p}\,\dfrac{\Gamma,A}{?\Gamma,!A} \quad \Delta,A^{\perp},?A^{\perp}}{?\Gamma,\Delta,A^{\perp}}}{\Gamma,?\Gamma,\Delta}}{\overline{?\Gamma,\Delta}}$$

FIGURE 5. Exponential cut-elimination steps in PLL_2.

$$\mathsf{cut}\,\dfrac{\forall\,\dfrac{\Gamma,A}{\Gamma,\forall X.A} \quad \exists\,\dfrac{\Delta,A^{\perp}[B/X]}{\Delta,\exists X.A^{\perp}}}{\Gamma,\Delta} \to_{\mathsf{cut}} \mathsf{cut}\,\dfrac{\Gamma,A[B/X] \quad \Delta,A^{\perp}[B/X]}{\Gamma,\Delta}$$

FIGURE 6. Second-order cut-elimination steps in PLL_2.

$$\mathsf{cut}\,\dfrac{\mathsf{r}\,\dfrac{\Gamma_1,A}{\Gamma,A} \quad A^{\perp},\Delta}{\Gamma,\Delta} \to_{\mathsf{cut}} \mathsf{r}\,\dfrac{\mathsf{cut}\,\dfrac{\Gamma_1,A \quad A^{\perp},\Delta}{\Gamma_1,\Delta}}{\Gamma,\Delta} \qquad \mathsf{cut}\,\dfrac{\mathsf{r}\,\dfrac{\Gamma_1,A \quad \Gamma_2}{\Gamma,A} \quad \Delta,A^{\perp}}{\Gamma,\Delta} \to_{\mathsf{cut}} \mathsf{r}\,\dfrac{\mathsf{cut}\,\dfrac{\Gamma_1,A \quad A^{\perp},\Delta}{\Gamma_1,\Delta} \quad \Gamma_2}{\Gamma,\Delta}$$

FIGURE 7. Commutative cut-elimination steps in PLL_2, where $\mathsf{r} \neq \mathsf{cut}$.

Theorem 9. *For every $\mathcal{D} \in \mathsf{PLL}_2$, there is a cut-free $\mathcal{D}' \in \mathsf{PLL}_2$ such that $\mathcal{D} \to^*_{\mathsf{cut}} \mathcal{D}'$.*

Sketch of proof. We associate with any derivation $\mathcal{D}$ in PLL_2 a derivation $\mathcal{D}^{♠}$ in MELL_2 sequent calculus. Thanks to additional commutative cut-elimination steps, we prove that cut-elimination in MELL_2 rewrites $\mathcal{D}^{♠}$ to the translation of a cut-free derivation in PLL_2. The termination of cut-elimination in PLL_2 follows from the result in MELL_2 [34]. Details for the propositional fragment are in [35], and extend to the second-order setting in a straightforward way. □

A byproduct of our grand tour diagram in Figure 1 is that PLL_2 represents exactly the class of functions in **FP** (Theorem 12). To see this, we introduce a rather permissive notion of representability for PLL_2, along the lines of [31].

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 (total) function $f : T_1 \times \ldots \times T_n \to T$ is **representable in PLL_2** if, for some $\mathbf{T_1}, \ldots, \mathbf{T_n}, \mathbf{T}$ representing $T_1, \ldots, T_n, T$ respectively, there is a derivation

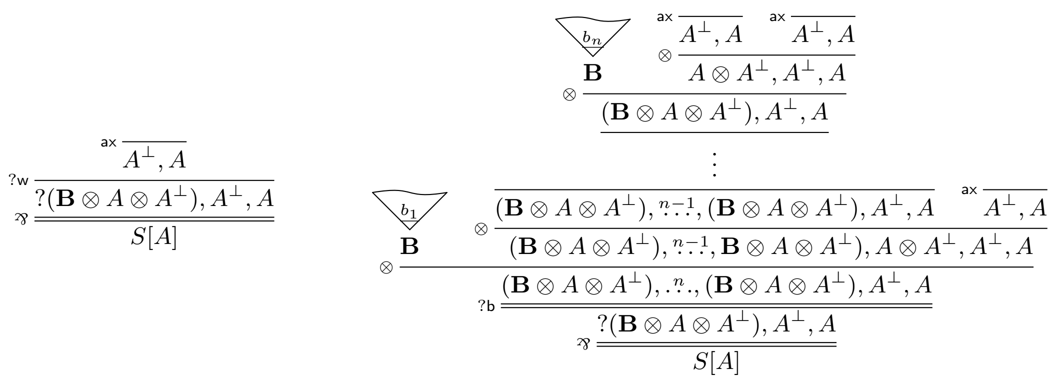
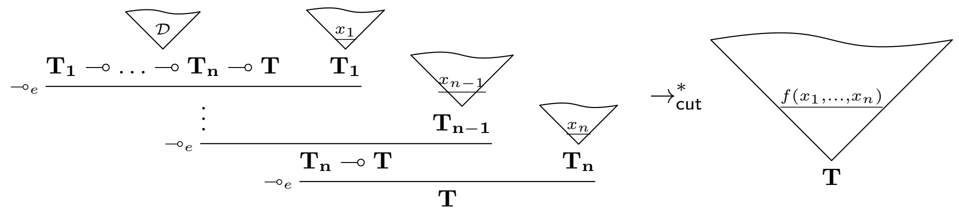

ax
$A^\perp, A$
?w
$?(\mathbf{B} \otimes A \otimes A^\perp), A^\perp, A$
⅋
$S[A]$
b_n
$\mathbf{B}$
$A \otimes A^\perp, A^\perp, A$
$(\mathbf{B} \otimes A \otimes A^\perp), A^\perp, A$
b_1
$(\mathbf{B} \otimes A \otimes A^\perp), \overset{n-1}{\ldots}, (\mathbf{B} \otimes A \otimes A^\perp), A^\perp, A$
$(\mathbf{B} \otimes A \otimes A^\perp), \overset{n-1}{\ldots}, \mathbf{B} \otimes A \otimes A^\perp), A \otimes A^\perp, A^\perp, A$
$(\mathbf{B} \otimes A \otimes A^\perp), \overset{n}{\ldots}, (\mathbf{B} \otimes A \otimes A^\perp), A^\perp, A$
?b

FIGURE 8. Encoding of the empty string ε and of a boolean string $\underline{b_1 \cdots b_n} \in \{0,1\}^*$.

$\mathcal{D}$ of $\mathbf{T_1} \multimap \ldots \multimap \mathbf{T_n} \multimap \mathbf{T}$ in PLL_2 such that, for all $x_1 \in T_1, \ldots, x_n \in T_n$:


$\mathcal{D}$
$\mathbf{T_1} \multimap \ldots \multimap \mathbf{T_n} \multimap \mathbf{T}$
x_1
$\mathbf{T_1}$
x_{n-1}
$\mathbf{T_{n-1}}$
x_n
$\mathbf{T_n}$
$\mathbf{T_n} \multimap \mathbf{T}$
$\mathbf{T}$
$\multimap_e$
$\rightarrow^*_{\mathsf{cut}}$
$f(x_1,\ldots,x_n)$

where $\multimap_e \dfrac{\Gamma, A \multimap B \quad \Delta, A}{\Gamma, \Delta, B}$ is shorthand notation for:

$$\mathsf{cut}\,\dfrac{\Gamma, A \multimap B \qquad \otimes\,\dfrac{\Delta, A \qquad \mathsf{ax}\,\dfrac{}{B^\perp, B}}{\Delta, (A \multimap B)^\perp, B}}{\Gamma, \Delta, B}$$

We denote with $\underline{f}$ a derivation representing f.

Example 11. The set of booleans $\mathbb{B} = \{\mathbf{0}, \mathbf{1}\}$ is represented in PLL_2 by the formula $\mathbf{B} = \forall X.(X^\perp ⅋ X^\perp) ⅋ (X \otimes X)$ thanks to the derivations $\underline{\mathbf{0}}$ and $\underline{\mathbf{1}}$ shown in Figure 3. Boolean strings are encoded 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., $\mathbf{S}[A] := !(\mathbf{B} \multimap A \multimap A) \multimap A \multimap A$, for any formula A. We write $\mathbf{S}[]$ to denote $\mathbf{S}[A]$ for some A. The encoding of boolean strings is as in Figure 8.

Akin to light linear logic [21, 36, 37], the exponential rules of PLL_2 are weaker than those in MELL_2: the usual promotion rule is replaced by $\mathsf{f!p}$ (*functorial promotion*), and the usual contraction and dereliction rules by $?\mathsf{b}$. 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).

This allows us to interpret computationally these weaker exponentials in terms of streams, as well as to control the complexity of cut-elimination [26, 27], which can be polynomially bounded. As a consequence, we obtain the following characterisation theorem for PLL_2:

Theorem 12 (PLL_2 characterises $\mathbf{FP}$)**.** *A function f is in $\mathbf{FP}$ iff f is representable in PLL_2.*

On the other hand, it is easy to show that $\mathsf{MELL}_2 = \mathsf{PLL}_2 + \text{digging}$: if we add the digging formula as an axiom to the set of rules in Figure 2, then the contraction formula becomes provable, and the obtained proof system coincides with MELL_2.

Remark 13. 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 as discussed in more detail in Section 6.2 (see Remark 77).

4. Non-wellfounded Parsimonious Linear Logic

In linear logic, a formula $!A$ is interpreted as the availability of A at will. This intuition still holds in PLL_2. Indeed, the Curry-Howard correspondence interprets rule $\mathsf{f!p}$ introducing the modality $!$ as an operator taking a derivation $\mathcal{D}$ of A and creating a (infinite) *stream* $(\mathcal{D}, \mathcal{D}, \dots, \mathcal{D}, \dots)$ of copies of the proof $\mathcal{D}$. Each element of the stream is accessed via the cut-elimination step $\mathsf{f!p}$-vs-$\mathsf{?b}$ in Figure 5: rule $\mathsf{?b}$ is interpreted as an operator *popping* one copy of $\mathcal{D}$ out of the stream. Pushing these ideas further, Mazza [26] introduced *parsimonious logic* $\mathbf{PL}$, a type system (comprising rules $\mathsf{f!p}$ and $\mathsf{?b}$) characterizing the logspace decidable problems.

Mazza and Terui then introduced in [27] another type system, $\mathbf{nuPL}_{\forall\ell}$, based on parsimonious logic and capturing the complexity class $\mathbf{P}/\mathsf{poly}$ (i.e., the problems decidable by polynomial size families of boolean circuits [20]). Their system is endowed with a *non-uniform* version of the functorial promotion, which takes a finite set of proofs $\mathcal{D}_1, \dots, \mathcal{D}_n$ of A and a (possibly non-recursive) function $f \colon \mathbb{N} \to \{1, \dots, n\}$ as premises, and constructs a proof of $!A$ modelling the stream $(\mathcal{D}_{f(0)}, \mathcal{D}_{f(1)}, \dots, \mathcal{D}_{f(n)}, \dots)$. This typing rule is used to encode the so-called *advices* for Turing machines, an essential step to show completeness for $\mathbf{P}/\mathsf{poly}$.

In a similar vein, we can endow PLL_2 with a non-uniform version of $\mathsf{f!p}$ called **infinitely branching promotion** ($\mathsf{ib!p}$), which constructs a stream $(\mathcal{D}_0, \mathcal{D}_1, \dots, \mathcal{D}_n, \dots)$ with finite support, i.e., made of *finitely* many distinct derivations (of the same conclusion):[3]

$$(1) \qquad \mathsf{ib!p}\,\frac{\overset{\mathcal{D}_0}{\Gamma, A} \qquad \overset{\mathcal{D}_1}{\Gamma, A} \qquad \cdots \qquad \overset{\mathcal{D}_n}{\Gamma, A} \qquad \cdots}{?\Gamma, !A}\ \{\mathcal{D}_i \mid i \in \mathbb{N}\} \text{ is finite} \quad \Bigg| \quad \mathsf{!w}\,\frac{}{!A} \qquad \mathsf{!b}\,\frac{\Gamma, A \quad \Delta, !A}{\Gamma, \Delta, !A}$$

The side condition on $\mathsf{ib!p}$ provides a proof theoretic counterpart to the function $f \colon \mathbb{N} \to \{1, \dots, n\}$ in $\mathbf{nuPL}_{\forall\ell}$. Clearly, $\mathsf{f!p}$ is subsumed by the rule $\mathsf{ib!p}$, as it corresponds to the special (uniform) case where $\mathcal{D}_i = \mathcal{D}_{i+1}$ for all $i \in \mathbb{N}$.

[3]Rule $\mathsf{ib!p}$ is reminiscent of the ω-rule used in (first-order) Peano arithmetic to derive formulas of the form $\forall x \phi$ that cannot be proven in a uniform way.

Definition 14. We define the set of rules $\mathsf{nuPLL}_2 := \{\mathsf{ax}, \otimes, \parr, 1, \bot, \mathsf{cut}, ?\mathsf{b}, ?\mathsf{w}, \mathsf{ib!p}, \forall, \exists\}$. We also denote by nuPLL_2 the set of derivations over the rules in nuPLL_2.[4]

There are some notable differences between nuPLL_2 and Mazza and Terui's system $\mathbf{nuPL}_{\forall\ell}$ [27]. As opposed to nuPLL_2, $\mathbf{nuPL}_{\forall\ell}$ is designed as an intuitionistic (type) system. Furthermore, to achieve completeness for $\mathbf{P}/\mathsf{poly}$, the latter is endowed with the co-absorption ($!\mathsf{b}$) and co-weakening ($!\mathsf{w}$) rules displayed in (1).

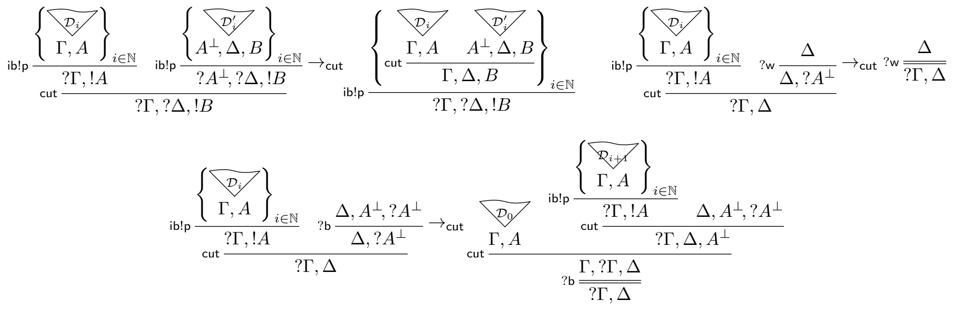


FIGURE 9. Exponential cut-elimination steps in nuPLL_2.

Cut-elimination steps for nuPLL_2 are in Figures 4, 7 and 9. In particular, the step $\mathsf{ib!p}$-vs-$?\mathsf{b}$ in Figure 9 *pops* the first premise $\mathcal{D}_0$ of $\mathsf{ib!p}$ out of the stream $(\mathcal{D}_0, \mathcal{D}_1, \dots, \mathcal{D}_n, \dots)$.

The notion of representability from Definition 10 extends to nuPLL_2 in the obvious way.

A byproduct of our gran tour diagram in Figure 1 is the following characterisation result:

Theorem 15. *A function f is in $\mathbf{FP}/\mathsf{poly}$ iff it is representable in nuPLL_2.*

4.1. **From infinitely branching proofs to non-wellfounded proofs.** In this paper we explore a dual approach to $\mathbf{nuPL}_{\forall\ell}$ (and nuPLL_2): instead of considering (wellfounded) derivations with infinite branching, we consider (non-wellfounded) coderivations with finite branching. To this end, the infinitary rule $\mathsf{ib!p}$ of nuPLL_2 is replaced by the binary rule below, called **conditional promotion** (c!p):

$$\text{(2)} \qquad \mathsf{c!p}\,\frac{\Gamma, A \quad ?\Gamma, !A}{?\Gamma, !A}$$

Definition 16. We define the set of rules $\mathsf{PLL}_2^\infty := \{\mathsf{ax}, \otimes, \parr, 1, \bot, \mathsf{cut}, ?\mathsf{b}, ?\mathsf{w}, \mathsf{c!p}, \forall, \exists\}$. We also denote by PLL_2^∞ the set of coderivations over the rules in PLL_2^∞.

In other words, PLL_2^∞ is the set of coderivations generated by the same rules as PLL_2, except that $\mathsf{f!p}$ is replaced by $\mathsf{c!p}$. From now on, we will only consider coderivations in PLL_2^∞.

Example 17. Figure 10 shows two non-wellfounded coderivations in PLL_2^∞: $\mathcal{D}_\bot$ (resp. $\mathcal{D}_?$) has an infinite branch of cut (resp. $?\mathsf{b}$) rules, and is (resp. is not) regular.

[4]To be rigorous, this requires a slight change in Definition 1: the tree labeled by a derivation in nuPLL_2 must be over $\mathbb{N}^\omega$ instead of $\{1,2\}^*$, in order to deal with infinitely branching derivations.

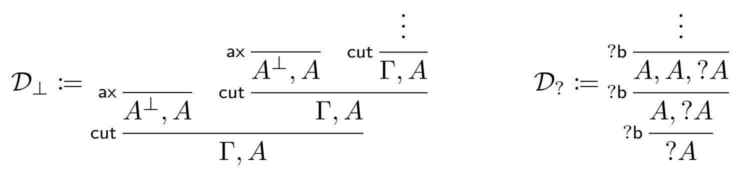
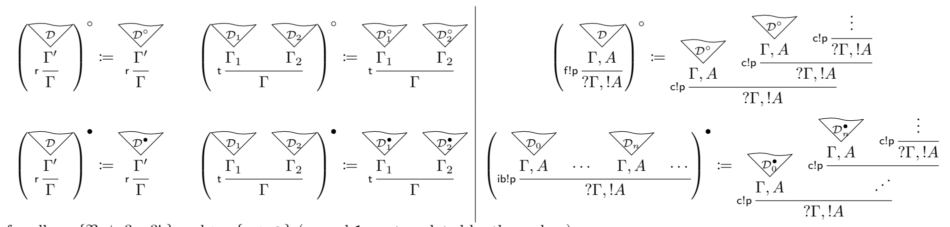


FIGURE 10. Two non-wellfounded and non-progressing coderivations in PLL_2^∞.



for all $\mathsf{r} \in \{⅋, \bot, ?\mathsf{w}, ?\mathsf{b}\}$ and $\mathsf{t} \in \{\mathsf{cut}, \otimes\}$ (ax and 1 are translated by themselves).

FIGURE 11. Translations $(\cdot)^\circ$ from PLL_2 to PLL_2^∞, and $(\cdot)^\bullet$ from nuPLL_2 to PLL_2^∞.

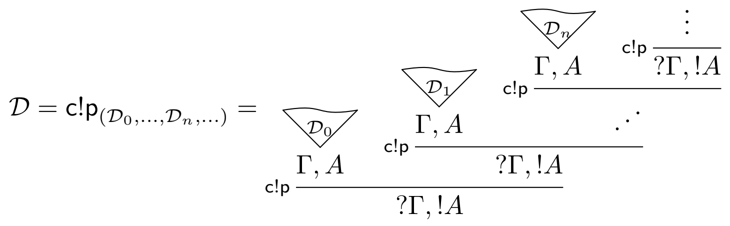


FIGURE 12. A non-wellfounded box in PLL_2^∞.

We can embed PLL_2 and nuPLL_2 into PLL_2^∞ via the conclusion-preserving translations $(\cdot)^\circ \colon \mathsf{PLL}_2 \to \mathsf{PLL}_2^\infty$ and $(\cdot)^\bullet \colon \mathsf{nuPLL}_2 \to \mathsf{PLL}_2^\infty$ defined in Figure 11 by induction on derivations: they map all rules to themselves except $\mathsf{f!p}$ and $\mathsf{ib!p}$, which are "unpacked" into non-wellfounded coderivations that iterate infinitely many times the rule $\mathsf{c!p}$.

An infinite chain of $\mathsf{c!p}$ rules (Figure 12) is a coderivation of PLL_2^∞ that is interesting on its own right.

Definition 18. A **non-wellfounded box** (nwb for short) is a coderivation $\mathcal{D}$ with an infinite branch $\{\epsilon, 2, 22, \dots\}$ (the **main branch** of $\mathcal{D}$) all labeled by $\mathsf{c!p}$ rules as in Figure 12, where $!A$ in the conclusion is the **principal formula** of $\mathcal{D}$, and $\mathcal{D}_0, \mathcal{D}_1, \dots$ are the **calls** of $\mathcal{D}$. We denote $\mathcal{D}$ by $\mathsf{c!p}_{(\mathcal{D}_0,\dots,\mathcal{D}_n,\dots)}$.

Let $\mathfrak{S} = \mathsf{c!p}_{(\mathcal{D}_0,\dots,\mathcal{D}_n,\dots)}$ be a nwb. We may write $\mathfrak{S}(i)$ to denote $\mathcal{D}_i$. We say that $\mathfrak{S}$ has **finite support** (resp. is **periodic** with **period** k) if $\{\mathfrak{S}(i) \mid i \in \mathbb{N}\}$ is finite (resp. if $\mathfrak{S}(i) = \mathfrak{S}(k+i)$ for any $i \in \mathbb{N}$). A coderivation $\mathcal{D}$ has **finite support** (resp. is **periodic**) if any nwb in $\mathcal{D}$ has finite support (resp. is periodic).

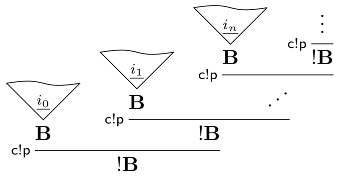
$$\mathsf{cut}\,\dfrac{\mathsf{c!p}\,\dfrac{\Gamma, A \quad ?\Gamma, !A}{?\Gamma, !A} \qquad \mathsf{c!p}\,\dfrac{A^\perp, \Delta, B \quad ?A^\perp, ?\Delta, !B}{?A^\perp, ?\Delta, !B}}{?\Gamma, ?\Delta, !B} \;\to_{\mathsf{cut}}\; \mathsf{c!p}\,\dfrac{\mathsf{cut}\,\dfrac{\Gamma, A \quad A^\perp, \Delta, B}{\Gamma, \Delta, B} \qquad \mathsf{cut}\,\dfrac{?\Gamma, !A \quad ?A^\perp, ?\Delta, !B}{?\Gamma, ?\Delta, !B}}{?\Gamma, ?\Delta, !B}$$

$$\mathsf{cut}\,\dfrac{\mathsf{c!p}\,\dfrac{\Gamma, A \quad ?\Gamma, !A}{?\Gamma, !A} \qquad ?\mathsf{w}\,\dfrac{\Delta}{\Delta, ?A^\perp}}{?\Gamma, \Delta} \;\to_{\mathsf{cut}}\; ?\mathsf{w}\,\dfrac{\Delta}{?\Gamma, \Delta} \qquad \mathsf{cut}\,\dfrac{\mathsf{c!p}\,\dfrac{\Gamma, A \quad ?\Gamma, !A}{?\Gamma, !A} \qquad ?\mathsf{b}\,\dfrac{\Delta, A^\perp, ?A^\perp}{\Delta, ?A^\perp}}{?\Gamma, \Delta} \;\to_{\mathsf{cut}}\; ?\mathsf{b}\,\dfrac{\mathsf{cut}\,\dfrac{\Gamma, A \qquad \mathsf{cut}\,\dfrac{?\Gamma, !A \quad \Delta, A^\perp, ?A^\perp}{?\Gamma, \Delta, A^\perp}}{\Gamma, ?\Gamma, \Delta}}{?\Gamma, \Delta}$$

FIGURE 13. Exponential cut-elimination steps for coderivations of PLL_2^∞.

Example 19. Consider the following nwb of the formula $!\mathbf{B}$, where $\mathbf{B}$ has at two distinct derivations $\underline{\mathbf{0}}$ and $\underline{\mathbf{1}}$ (Example 8), and $i_j \in \{\mathbf{0}, \mathbf{1}\}$ for all $j \in \mathbb{N}$.

(3) $\mathsf{c!p}_{(\underline{i_0}, \ldots, \underline{i_n}, \ldots)} =$



Thus $\mathsf{c!p}_{(\underline{i_0}, \ldots, \underline{i_n}, \ldots)}$ has finite support, as its only calls can be $\underline{\mathbf{0}}$ or $\underline{\mathbf{1}}$, and it is periodic if and only if so is the infinite sequence $(i_0, \ldots, i_n, \ldots) \in \{\mathbf{0}, \mathbf{1}\}^\omega$.

The *cut-elimination* steps $\to_{\mathsf{cut}}$ for PLL_2^∞ are in Figures 4, 7 and 13. Computationally, they allow the $\mathsf{c!p}$ rule to be interpreted as a *coinductive* definition of a stream of type $!A$ from a stream of the same type to which an element of type A is prepended. In particular, the cut-elimination step $\mathsf{c!p}$-vs-$?\mathsf{b}$ accesses the head of a stream: rule $?\mathsf{b}$ acts as a *popping* operator.

As a consequence, the nwb in Figure 12 constructs a stream $(\mathcal{D}_0, \mathcal{D}_1, \ldots, \mathcal{D}_n, \ldots)$ similarly to $\mathsf{ib!p}$ but, unlike the latter, all the $\mathcal{D}_i$'s may be pairwise distinct. The reader familiar with linear logic can see a nwb as a box with possibly *infinitely many* distinct contents (its calls), while usual linear logic boxes (and $\mathsf{f!p}$ in PLL_2) provide infinitely many copies of the *same* content.

Rules $\mathsf{f!p}$ in PLL_2 and $\mathsf{ib!p}$ in nuPLL_2 are mapped by $(\cdot)^\circ$ and $(\cdot)^\bullet$ into nwbs, which are non-wellfounded coderivations. Hence, the cut-elimination steps $\mathsf{f!p}$-vs-$\mathsf{f!p}$ in PLL_2 and $\mathsf{ib!p}$-vs-$\mathsf{ib!p}$ in nuPLL_2 can only be simulated by infinitely many cut-elimination steps in PLL_2^∞.

Note that $\mathcal{D}_\perp \in \mathsf{PLL}_2^\infty$ in Figure 10 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 so the cut-elimination theorem fails in PLL_2^∞.

The notion of representability in Definition 10 can be easily adapted to PLL_2^∞, essentially by generalising derivations to arbitrary coderivations.

4.2. **Consistency via a progressing condition.**

In a non-wellfounded setting such as PLL_2^∞, any sequent is provable. Indeed, the (non-wellfounded) coderivation $\mathcal{D}_\perp$ in Figure 10 shows that any non-empty sequent (in particular, any formula) is provable in PLL_2^∞.

The standard way to recover logical consistency in non-wellfounded proof theory is to introduce a global soundness condition on coderivations, called **progressing criterion**. In PLL_2^∞, this criterion relies on tracking occurrences of !-formulas in a coderivation.

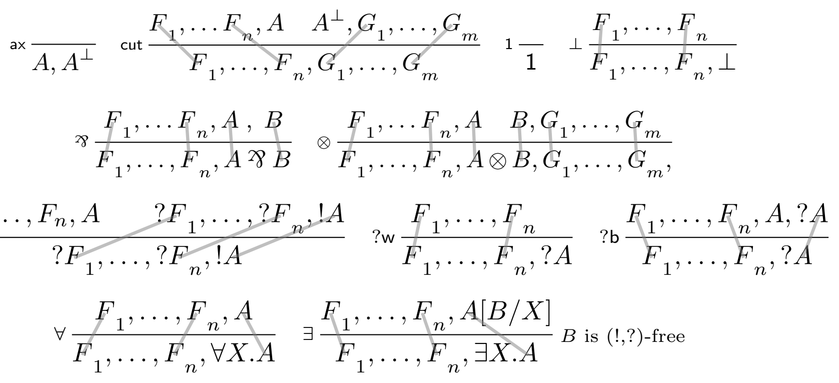


FIGURE 14. PLL_2^∞ rules: edges connect a formula in the conclusion with its parent(s) in a premise.

Definition 20. Let $\mathcal{D}$ be a coderivation in PLL_2^∞. It is **weakly progressing** if every infinite branch contains infinitely many right premises of $\mathsf{c!p}$-rules.

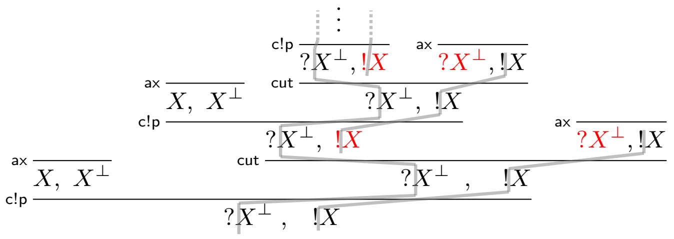
An occurrence of formula in a premise of a rule r is the **parent** of an occurrence of a formula in the conclusion if they are connected according to the edges depicted in Figure 14.

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 $\mathsf{c!p}$ for infinitely many $j \in I$.

$\mathcal{D}$ is **progressing** if every infinite branch contains a progressing !-thread. We define pPLL_2^∞ (resp. wpPLL_2^∞) as the set of progressing (resp. weakly progressing) coderivations in PLL_2^∞.

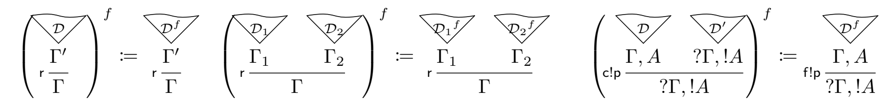
Remark 21. Clearly, any progressing coderivation is weakly progressing too, but the converse fails (Example 22), therefore $\mathsf{pPLL}_2^\infty \subsetneq \mathsf{wpPLL}_2^\infty$. Moreover, the main branch of any nwb contains by definition a progressing !-thread of its principal formula.

Example 22. Coderivations in Figure 10 are not weakly progressing (hence, 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 $\mathsf{c!p}$-rules. In contrast, the nwb $\mathsf{c!p}_{(\underline{i_0},\ldots,\underline{i_n},\ldots)}$ in Example 19 is progressing by Remark 21, since its main branch is the only infinite branch. Below, a regular, weakly progressing but not progressing coderivation ($!X$ in the conclusion of $\mathsf{c!p}$ is a cut-formula, so the branch $\{\epsilon, 2, 21, 212, 2121, \ldots\}$ is infinite but has no progressing !-thread).



Lemma 23. *Let Γ be a sequent. Then, $\vdash_{\mathsf{PLL}_2} \Gamma$ if and only if $\vdash_{\mathsf{wpPLL}_2^\infty} \Gamma$.*

Proof. Given $\mathcal{D} \in \mathsf{PLL}_2$, $\mathcal{D}^\bullet \in \mathsf{PLL}_2^\infty$ preserves the conclusion and is progressing, hence weakly progressing (see Remark 21). Conversely, given a weakly progressing coderivation $\mathcal{D}$, we define a derivation $\mathcal{D}^f \in \mathsf{PLL}_2$ with the same conclusion by applying, bottom-up, the translation:

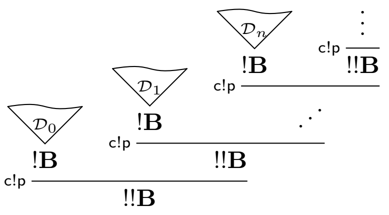


with $\mathsf{r} \neq \mathsf{c!p}$. Note that the derivation $\mathcal{D}^f$ is well-defined because $\mathcal{D}$ is weakly progressing. ☐

Corollary 24. *The empty sequent is not provable in wpPLL_2^∞ (and hence in pPLL_2^∞).*

Proof. If the empty sequent were provable in wpPLL_2^∞, then there would be a cut-free derivation $\mathcal{D} \in \mathsf{PLL}_2$ of the empty sequent by Lemma 23 and Theorem 9, but this is impossible since cut is the only rule in PLL_2 that could have the empty sequent in its conclusion. ☐

4.3. **Recovering (weak forms of) regularity.** The progressing criterion cannot capture the finiteness condition of the rule $\mathsf{ib!p}$ in the derivations in nuPLL_2. By means of example, consider the nwb below, which is progressive but cannot be the image of the rule $\mathsf{ib!p}$ via $(\cdot)^\bullet$ (see Figure 11) since $\{\mathcal{D}_i \mid i \in \mathbb{N}\}$ is infinite.

(4)


with $\mathcal{D}_i = \mathsf{c!p}_{(\underbrace{\underline{\mathbf{1}},\dots,\underline{\mathbf{1}}}_{i},\underline{\mathbf{0}},\dots)}$ for each $i \in \mathbb{N}$.

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

Remark 26. Regularity implies weak regularity and the converse fails as shown in Example 27 below, therefore $\mathsf{rPLL}_2^\infty \subsetneq \mathsf{wrPLL}_2^\infty$. Moreover, $\mathcal{D} \in \mathsf{PLL}_2^\infty$ is regular (resp. weakly regular) if and only if any nwb in $\mathcal{D}$ is periodic (resp. has finite support).

Example 27. Coderivations $\mathcal{D}_\perp$ and $\mathcal{D}_?$ in Figure 10 are not finitely expandable, as their infinite branch has infinitely many cut or $?\mathsf{b}$, but they are weakly regular, since they have no $\mathsf{c!p}$ rules. The coderivation in (4) is not weakly regular because $\{\mathcal{D}_i \mid i \in \mathbb{N}\}$ is infinite.

An example of a weakly regular but not regular coderivation is the nwb $\mathsf{c!p}_{(\underline{i_0},\dots,\underline{i_n},\dots)}$ in Example 19 when the infinite sequence $(i_j)_{j\in\mathbb{N}} \in \{\mathbf{0},\mathbf{1}\}^\omega$ is not periodic: in each rule $\mathsf{c!p}$ there, the left premise can only be $\underline{\mathbf{0}}$ or $\underline{\mathbf{1}}$ (so the nwb is weakly regular), but the right premise is a distinct coderivation (so the nwb is not regular). Moreover, that nwb is finitely expandable since it contains no $?\mathsf{b}$ or cut.

By inspecting the steps in Figures 4, 7 and 13, we prove the following preservations.

Proposition 28. *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 $\mathcal{D}' \in \mathsf{X}$.*

Proof. By inspection of the cut-elimination steps defined in Figures 4, 7 and 13. ☐

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

Definition 29. Let $\mathcal{D} \in \mathsf{PLL}_2^\infty$. 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 contains 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 30. All calls of a nwb have the same nesting level. Moreover, each of the sequents of its main branch have nesting level 0.

Cut-elimination $\to_{\mathsf{cut}}$ on PLL_2^∞ enjoys the following property.

Lemma 31. *Let $\mathcal{D}, \mathcal{D}' \in \mathsf{PLL}_2^\infty$. If $\mathcal{D} \to_{\mathsf{cut}} \mathcal{D}'$ then $\mathbf{d}(\mathcal{D}) \geq \mathbf{d}(\mathcal{D}')$.*

Proof. By inspection of the cut-elimination steps in Figures 4, 7 and 13. ☐

Lemma 32. *If $\mathcal{D} \in \mathsf{pPLL}_2^\infty$ then $\mathbf{d}(\mathcal{D}) \in \mathbb{N}$.*

Proof. If $\mathcal{D}$ had infinite depth, there would exist an infinite branch that goes left at $\mathsf{c!p}$ infinitely often. This branch cannot contain a (progressing) !-thread. ☐

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 $(\cdot)^\bullet$.

Proposition 33.

(1) If $\mathcal{D} \in \mathsf{PLL}_2$ (resp. $\mathcal{D} \in \mathsf{nuPLL}_2$) with conclusion Γ, then $\mathcal{D}^\circ \in \mathsf{rPLL}_2^\infty$ (resp. $\mathcal{D}^\bullet \in \mathsf{wrPLL}_2^\infty$) with conclusion Γ, and every $\mathsf{c!p}$ in $\mathcal{D}^\circ$ (resp. $\mathcal{D}^\bullet$) belongs to a nwb.

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

Proof. Item 1 is proven by straightforward induction on $\mathcal{D} \in \mathsf{PLL}_2$ (resp. $\mathcal{D} \in \mathsf{nuPLL}_2$). Concerning Item 2, by Lemma 32 we have $\mathbf{d}(\mathcal{D}) \in \mathbb{N}$. We can then prove the statement by induction on $\mathbf{d}(\mathcal{D})$. ☐

Progressing and weak progressing coincide in finite expandable coderivations.

Lemma 34. *Let $\mathcal{D} \in \mathsf{PLL}_2^\infty$ be finitely expandable. If $\mathcal{D} \in \mathsf{wpPLL}_2^\infty$ then any infinite branch contains the principal branch of a* nwb*. Moreover, $\mathcal{D} \in \mathsf{pPLL}_2^\infty$ iff $\mathcal{D} \in \mathsf{wpPLL}_2^\infty$.*

Proof. Let $\mathcal{D} \in \mathsf{wpPLL}_2^\infty$ 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 $?\mathsf{b}$ with height greater than h. Moreover, by weakly progressing there is an infinite sequence $h \leq h_0 < h_1 < \ldots < h_n < \ldots$ 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 either the conclusion of either a $?\mathsf{w}$ or a $\mathsf{c!p}$ (with right premise $?\Gamma_i, !A_i$). So, there is a k large enough such that, for any $i \geq k$, 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 $\mathcal{D} \in \mathsf{pPLL}_2^\infty$. By Remark 21, $\mathsf{pPLL}_2^\infty \subseteq \mathsf{wpPLL}_2^\infty$. □

Proposition 35. *It is* **NL***-decidable if a regular coderivation is in rPLL_2^∞.*

Proof. A regular coderivation is represented by a finite cyclic graph. By Lemma 34 checking progressiveness comes down to checking that no branch has infinitely many occurrences of a particular rule, which in turn reduces to checking acyclicity for this graph (see [38]). We conclude since checking acyclicity is a well-known **coNL** problem, and **coNL** = **NL**[20]. □

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 19.

4.4. Simulation results.

We conclude this section by establishing a simulation result showing that any function on boolean strings representable in nuPLL_2 is also representable in wrPLL_2^∞, and similarly for PLL_2 and rPLL_2^∞.

Theorem 36 (Simulation)**.** *Let $f : (\{\mathbf{0},\mathbf{1}\}^*)^n \to \{\mathbf{0},\mathbf{1}\}^*$.*

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

Proof. We consider the translations $(\cdot)^\circ$ from PLL_2 to PLL_2^∞, and $(\cdot)^\bullet$ from nuPLL_2 to PLL_2^∞ in Figure 11. On the one hand, by Proposition 33 we have:

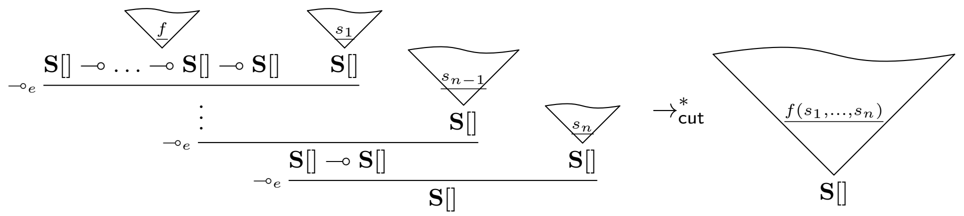
(1) If $\mathcal{D} \in \mathsf{nuPLL}_2$ then $\mathcal{D}^\bullet \in \mathsf{wrPLL}_2^\infty$.
(2) If $\mathcal{D} \in \mathsf{PLL}_2$ then $\mathcal{D}^\circ \in \mathsf{rPLL}_2^\infty$.

On the other hand, it is easy to check that those translations satisfy the following properties:

(4) If $\mathcal{D}_1 \in \mathsf{nuPLL}_2$ and $\mathcal{D}_2$ is obtained from $\mathcal{D}_1$ by applying a cut-elimination step different from ($\mathsf{f!p}$-vs-$\mathsf{f!p}$) and ($\mathsf{ib!p}$-vs-$\mathsf{ib!p}$) then $\mathcal{D}_1^\bullet \to_{\mathsf{cut}} \mathcal{D}_2^\bullet$.
(5) If $\mathcal{D}_1 \in \mathsf{nuPLL}_2$ and $\mathcal{D}_2$ is obtained from $\mathcal{D}_1$ by applying a cut-elimination step different from ($\mathsf{f!p}$-vs-$\mathsf{f!p}$) then $\mathcal{D}_1^\circ \to_{\mathsf{cut}} \mathcal{D}_2^\circ$.

Notice that the rules $\mathsf{f!p}$ and $\mathsf{ib!p}$ are mapped by $(\cdot)^\circ$ and $(\cdot)^\bullet$ to a bottomless sequence of rules. As a consequence, applications of the rules ($\mathsf{f!p}$-vs-$\mathsf{f!p}$) and ($\mathsf{ib!p}$-vs-$\mathsf{ib!p}$) can only be simulated by infinitely many cut-elimination steps in PLL_2^∞.

Now, let $\underline{f} : \mathbf{S}[] \multimap \overset{n \geq 0}{\cdots} \multimap \mathbf{S}[] \multimap \mathbf{S}[]$ represent $f : (\{\mathbf{0},\mathbf{1}\}^*)^n \to \{\mathbf{0},\mathbf{1}\}^*$ in nuPLL_2 (the case for PLL_2 is similar), and let $s_1, \ldots, s_n \in \{\mathbf{0},\mathbf{1}\}^*$. By hypothesis, we have:



Let $\sigma \coloneqq \mathcal{D} = \mathcal{D}_0 \to_{\mathsf{cut}} \mathcal{D}_1 \to_{\mathsf{cut}} \ldots \to_{\mathsf{cut}} \mathcal{D}_n = \underline{f(s_1, \ldots, s_n)}$ be the above cut-elimination sequence, where $\underline{f(s_1, \ldots, s_n)}$ is cut-free by definition. We now establish the following fact about cut-elimination in nuPLL_2:

(1) The cut-elimination steps for ($\mathsf{f!p}$-vs-$\mathsf{f!p}$) and ($\mathsf{ib!p}$-vs-$\mathsf{ib!p}$) commute with any other cut-elimination step;
(2) If $\mathcal{D}$ has !-free conclusion then either it is cut-free or it has a cut rule different from ($\mathsf{f!p}$-vs-$\mathsf{f!p}$) and ($\mathsf{ib!p}$-vs-$\mathsf{ib!p}$).

Item 1 is straightforward by inspecting the cut-elimination rules for nuPLL_2. Concerning Item 2, let us suppose towards contradiction that there is a derivation $\mathcal{D} \in \mathsf{nuPLL}_2$ whose cut rules are all with shape ($\mathsf{f!p}$-vs-$\mathsf{f!p}$) and ($\mathsf{ib!p}$-vs-$\mathsf{ib!p}$). Moreover, let us suppose that $\mathcal{D}$ is not cut-free, and let h be the smallest height of a cut. By assumption, this cut contains a !-formula in its conclusion. Now, by inspecting the rules of nuPLL_2 it is easy to see that any rule distinct from cut has a !-formula in its conclusion whenever one of its premises has (recall that instantiation in the $\exists$ rule requires !-freeness). Since no cut rule has height strictly smaller than h it follows that the conclusion of $\mathcal{D}$ contains a !-formula, contradicting our assumption.

Now, the two facts above allow us to conclude the existence of a cut-elimination sequence from $\mathcal{D}$ to $\underline{f(s_1, \ldots, s_n)}$ free of reduction steps for ($\mathsf{f!p}$-vs-$\mathsf{f!p}$) and ($\mathsf{ib!p}$-vs-$\mathsf{ib!p}$), so that we can conclude by applying Item 1, Item 2, and Item 4. Indeed, by Item 1, we can rewrite σ into another sequence σ' where all such cut-elimination steps are postponed. In particular, this means that there is a derivation $\mathcal{D}'$ such that $\mathcal{D} = \mathcal{D}'_0 \to_{\mathsf{cut}} \ldots \mathcal{D}'_k$ and $\mathcal{D}'_k \to_{\mathsf{cut}} \ldots \to_{\mathsf{cut}} \mathcal{D}'_n = \underline{f(s_1, \ldots, s_n)}$, for some $0 \leq k \leq n$ such that:

- $\mathcal{D} = \mathcal{D}'_0 \to_{\mathsf{cut}} \ldots \mathcal{D}'_k$ is free of cut-elimination steps ($\mathsf{f!p}$-vs-$\mathsf{f!p}$) and ($\mathsf{ib!p}$-vs-$\mathsf{ib!p}$),
- $\mathcal{D}'_k \to_{\mathsf{cut}} \ldots \to_{\mathsf{cut}} \mathcal{D}'_n = \underline{f(s_1, \ldots, s_n)}$ contains only cut-elimination steps for ($\mathsf{f!p}$-vs-$\mathsf{f!p}$) and ($\mathsf{ib!p}$-vs-$\mathsf{ib!p}$).

Since $\underline{f(s_1, \ldots, s_n)}$ is cut-free, we have $k = n$ by Item 2. □

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 37. We define the set of rules $\mathsf{oPLL}_2^\infty \coloneqq \mathsf{PLL}_2^\infty \cup \{\mathsf{hyp}\}$, where $\mathsf{hyp} \coloneqq$ $\mathsf{hyp}\,\dfrac{}{\Gamma}$ for any sequent Γ.[5] We will also refer to oPLL_2^∞ as the set of coderivations

[5]Previously introduced notions and definitions on coderivations extend to open coderivations in the obvious way, e.g. the global conditions Definition 20 and Definition 25, as well as the cut-elimination relation $\to_{\mathsf{cut}}$.

over oPLL_2^∞, which we call **open coderivations**. An open coderivation is **normal** if no cut-elimination step can be applied to it, that is, if one premise of each cut is a hyp. An **open derivation** is a derivation in oPLL_2^∞.

Definition 38. Let $\mathcal{D}$ be an open coderivation, $\mathcal{V} = \{\nu_1, \dots, \nu_n\} \subseteq \{1,2\}^*$ be a finite set of mutually incomparable (w.r.t. the prefix order) nodes of $\mathcal{D}$:

- Let $\{\mathcal{D}'_i\}_{1 \leq i \leq n}$ be a set of open coderivations where $\mathcal{D}'_i$ has the same conclusion as the subderivation $\mathcal{D}_{\nu_i}$ of $\mathcal{D}$. We denote by $\mathcal{D}(\mathcal{D}'_1/\nu_1, \dots, \mathcal{D}'_n/\nu_n)$, the open coderivation obtained by replacing each $\mathcal{D}_{\nu_i}$ with $\mathcal{D}'_i$.
- The **pruning** of $\mathcal{D}$ over $\mathcal{V}$ is the open coderivation $\lfloor \mathcal{D} \rfloor_{\mathcal{V}} = \mathcal{D}(\mathsf{hyp}/\nu_1, \dots, \mathsf{hyp}/\nu_n)$. If $\mathcal{D}$ and $\mathcal{D}'$ are two open coderivations, then we say that $\mathcal{D}$ is an **approximation** of $\mathcal{D}'$ (noted $\mathcal{D} \preceq \mathcal{D}'$) iff $\mathcal{D} = \lfloor \mathcal{D}' \rfloor_{\mathcal{V}}$ for some $\mathcal{V} \subseteq \{1,2\}^*$. An approximation is **finite** if it is an open derivation.

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

Proposition 39 (Cubic bound on finite approxmations)**.** *Let Γ be a sequent. If $\mathcal{D}$ be a finite approximation and*

$$\mathcal{D} = \mathcal{D}_0 \rightarrow_{\mathsf{cut}} \cdots \rightarrow_{\mathsf{cut}} \mathcal{D}_n$$

then $n \in \mathcal{O}(|\mathcal{D}|^3)$ and $|\mathcal{D}_i| \in \mathcal{O}(|\mathcal{D}|^3)$ for any $i \in \{0, \dots, n\}$. If moreover the reduction sequence is maximal then $\mathcal{D}_n$ is cut free.

Proof. Let w be number of ?-formulas Γ and k be the number of $\mathsf{c!p}$ rules in $\mathcal{D}$. We define the **weight of** $\mathcal{D}$ as $\mathsf{W}(\mathcal{D}) \coloneqq |\mathcal{D}| + wk$ and $\mathsf{H}(\mathcal{D})$ as the sum of the heights of all subproofs of $\mathcal{D}$ whose root is an application of the cut rule. Commutative cut-elimination steps $\mathcal{D} \rightarrow_{\mathsf{cut}} \mathcal{D}'$ satisfy $\mathsf{W}(\mathcal{D}') = \mathsf{W}(\mathcal{D})$ and $\mathsf{H}(\mathcal{D}') < \mathsf{H}(\mathcal{D})$; For non-commutative cut-elimination steps we have $\mathsf{W}(\mathcal{D}') < \mathsf{W}(\mathcal{D})$. Since the lexicograophic order over the pairs $\langle \mathsf{W}(\mathcal{D}), \mathsf{H}(\mathcal{D}) \rangle$ is wellfounded, we conclude that there cannot be an infinite reduction path starting from $\mathcal{D}$.

Now let $\mathcal{D}'$ be a normal derivation such that $\mathcal{D} \rightarrow_{\mathsf{cut}}^n \mathcal{D}'$. The number $n_{p,a}$ of principal cut-elimination steps is bounded by $\mathsf{W}(\mathcal{D})$. 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:

$$\begin{aligned} n = n_{p,a} + \textstyle\sum_{i=1}^{n_{p,a}} n_c^i \leq n_{p,a} + n_{p,a} \left(\max_i \{n_c^i + 1\}\right) \leq \\ \leq n_{p,a} \left(\max_i \{n_c^i + 1\}\right) \leq \mathsf{W}(\mathcal{D}) \cdot (\mathsf{W}(\mathcal{D})^2 + 1) \quad \leq 2\mathsf{W}(\mathcal{D})^3 \end{aligned}$$

We conclude since $\mathsf{W}(\mathcal{D}) \in \mathcal{O}(|\mathcal{D}|)$. □

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

Proposition 40. *If $\mathcal{D} \in \mathsf{pPLL}_2^\infty$ is finitely expandable, then there is a finite set $\mathcal{V} \subseteq \{1,2\}^*$ of nodes of $\mathcal{D}$ such that $\lfloor \mathcal{D} \rfloor_{\mathcal{V}}$ is a open derivation and each $v \in \mathcal{V}$ is the root of a nwb in $\mathcal{D}$.*

Proof. By Lemma 34, there is a set $\mathcal{V}$ of nodes of $\mathcal{D}$ such that: (i) each node in $\mathcal{V}$ is the root of a nwb, and (ii) any infinite branch of $\mathcal{D}$ contains a node in $\mathcal{V}$. Thus, $\lfloor \mathcal{D} \rfloor_{\mathcal{V}}$ must be finite by weak König's lemma, and so is $\mathcal{V}$. □

Definition 41. Let $\mathcal{D} \in \mathsf{pPLL}_2^\infty$ be finitely expandable. The **decomposition** of $\mathcal{D}$ is the (unique) set of nodes $\mathsf{border}(\mathcal{D}) = \{\nu_1, \dots, \nu_k\}$ with $k \in \mathbb{N}$ such that $\mathcal{D}_{\nu_i}$ is a nwb for all $i \in \{1, \dots, k\}$ and $\mathsf{base}(\mathcal{D}) \coloneqq \lfloor \mathcal{D} \rfloor_{\mathsf{border}(\mathcal{D})}$ is a minimal (w.r.t. $\preceq$) finite approximation.

5. 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 [39].

The relational semantics interprets the exponentials by finite multisets, denoted by brackets, e.g., $[x_1, \dots, x_n]$; $+$ 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 42 (Reflexive object)**.** We define $D \coloneqq \bigcup_{n\in\mathbb{N}} D_n$, where D_n is defined by induction as follows:

$$\begin{array}{rcl} D_0 & \coloneqq & \{*\} \\ D_{n+1} & \coloneqq & D_0 \cup (D_n \times D_n) \cup \mathcal{M}_f(D_n) \end{array}$$

Definition 43. We associate with each formula A a **set** $\{\{A\}\}$ defined as follows:

$$\begin{array}{rclcrcl} \{\{X\}\} & \coloneqq & D & \quad & \{\{A \otimes B\}\} & \coloneqq & \{\{A\}\} \times \{\{B\}\} \\ \{\{1\}\} & \coloneqq & \{*\} & & \{\{!A\}\} & \coloneqq & \mathcal{M}_f(\{\{A\}\}) \\ \{\{A^\perp\}\} & \coloneqq & \{\{A\}\} & & \{\{\forall X.A\}\} & \coloneqq & \{\{A\}\} \end{array}$$

where D is as in Definition 42. For a sequent $\Gamma = A_1, \dots, A_n$, we set $\{\{\Gamma\}\} \coloneqq \{\{A_1 \parr \cdots \parr A_n\}\}$.

Given oPLL_2^∞ with conclusion Γ, we set $\{\{\mathcal{D}\}\} \coloneqq \bigcup_{n\geq 0} \{\{\mathcal{D}\}\}_n \subseteq \{\{\Gamma\}\}$, where $\{\{\mathcal{D}\}\}_0 = \varnothing$ and, for all $i \in \mathbb{N} \setminus \{0\}$, $\{\{\mathcal{D}\}\}_i$ is defined inductively according to Figure 15.

Example 44. For the coderivations $\mathcal{D}_\perp$ and $\mathcal{D}_?$ in Figure 10, $\{\{\mathcal{D}_\perp\}\} = \{\{\mathcal{D}_?\}\} = \varnothing$. For the derivations $\underline{\mathbf{0}}$ and $\underline{\mathbf{1}}$ in Figure 3, $\{\{\underline{\mathbf{0}}\}\} = \{((x,y),(x,y)) \mid x \in D\}$ and $\{\{\underline{\mathbf{1}}\}\} = \{((x,y),(y,x)) \mid x, y \in D\}$. For the coderivation $\mathsf{c!p}_{(\underline{i_0}, \dots, \underline{i_n}, \dots)}$ in Example 19 (with $i_j \in \{\mathbf{0}, \mathbf{1}\}$ for all $j \in \mathbb{N}$), the relation $\left\{\left\{\mathsf{c!p}_{(\underline{i_0}, \dots, \underline{i_n}, \dots)}\right\}\right\}$ is defined as the set of all multisets of the form $[((x_1,y_1),(z_1,w_1)), \dots, ((x_n,y_n),(z_n,w_n))]$ with $n \in \mathbb{N}$ and $x_i, y_i, z_i, w_i \in D$ such that $x_i = w_i$ and $y_i = z_i$ whenever $\underline{i} = \underline{\mathbf{1}}$, and such that $x_i = z_i$ and $y_i = w_i$ whenever $\underline{i} = \underline{\mathbf{0}}$.

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

Theorem 45 (Soundness)**.** *Let $\mathcal{D} \in \mathsf{oPLL}_2^\infty$. If $\mathcal{D} \to_{\mathsf{cut}} \mathcal{D}'$, then $\{\{\mathcal{D}\}\} = \{\{\mathcal{D}'\}\}$.*

6. Characterisation results

In this section we prove the fundamental result of this paper:

Theorem 46.

$$\left\{\!\!\left\{ {\scriptstyle\mathsf{ax}}\,\dfrac{}{A, A^{\perp}} \right\}\!\!\right\}_n = \{\ (x,x) \mid x \in \{\!\{A\}\!\}\ \} \qquad \left\{\!\!\left\{ {\scriptstyle\mathsf{hyp}}\,\dfrac{}{\Gamma} \right\}\!\!\right\}_n = \varnothing$$

$$\left\{\!\!\left\{ {\scriptstyle\perp}\,\dfrac{\overset{\mathcal{D}'}{\Gamma}}{\Gamma, \perp} \right\}\!\!\right\}_n = \{\ (\vec{x}, *) \mid \vec{x} \in \{\!\{\mathcal{D}'\}\!\}_{n-1}\ \} \qquad \left\{\!\!\left\{ {\scriptstyle 1}\,\dfrac{}{1} \right\}\!\!\right\}_n = \{*\}$$

$$\left\{\!\!\left\{ {\scriptstyle\mathsf{cut}}\,\dfrac{\overset{\mathcal{D}'}{\Gamma, A} \qquad \overset{\mathcal{D}''}{\Delta, A^{\perp}}}{\Gamma, \Delta} \right\}\!\!\right\}_n = \left\{ (\vec{x}, \vec{y}) \ \middle|\ \exists z \in \{\!\{A\}\!\} \text{ s.t. } \begin{array}{c} (\vec{x}, z) \in \{\!\{\mathcal{D}'\}\!\}_{n-1} \\ \text{and} \\ (z, \vec{y}) \in \{\!\{\mathcal{D}''\}\!\}_{n-1} \end{array} \right\}$$

$$\left\{\!\!\left\{ {\scriptstyle\parr}\,\dfrac{\overset{\mathcal{D}'}{\Gamma, A, B}}{\Gamma, A \parr B} \right\}\!\!\right\}_n = \{\ (\vec{x}, (y,z)) \mid (\vec{x}, y, z) \in \{\!\{\mathcal{D}'\}\!\}_{n-1}\ \}$$

$$\left\{\!\!\left\{ {\scriptstyle\otimes}\,\dfrac{\overset{\mathcal{D}'}{\Gamma, A} \qquad \overset{\mathcal{D}''}{\Delta, B}}{\Gamma, \Delta, A \otimes B} \right\}\!\!\right\}_n = \left\{ (\vec{x}, \vec{y}, (x,y)) \ \middle|\ \begin{array}{c} (\vec{x}, x) \in \{\!\{\mathcal{D}'\}\!\}_{n-1} \\ \text{and} \\ (\vec{y}, y) \in \{\!\{\mathcal{D}''\}\!\}_{n-1} \end{array} \right\}$$

$$\left\{\!\!\left\{ {\scriptstyle ?\mathsf{w}}\,\dfrac{\overset{\mathcal{D}'}{\Gamma}}{\Gamma, ?A} \right\}\!\!\right\}_n = \{\ (\vec{x}, [\,]) \mid \vec{x} \in \{\!\{\mathcal{D}'\}\!\}_{n-1}\ \}$$

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

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

$$\left\{\!\!\left\{ {\scriptstyle\forall}\,\dfrac{\overset{\mathcal{D}'}{\Gamma, A}}{\Gamma, \forall X.A} \right\}\!\!\right\}_n = \left\{\!\!\left\{ {\scriptstyle\exists}\,\dfrac{\overset{\mathcal{D}'}{\Gamma, A[B/X]}}{\Gamma, \exists X.A} \right\}\!\!\right\}_n = \{\!\{\mathcal{D}'\}\!\}_{n-1}$$

FIGURE 15. Inductive definition of the set $\{\!\{\mathcal{D}\}\!\}_n$, for $n > 0$.

(1) $\mathsf{wrPLL}_2^{\infty} = \mathsf{nuPLL}_2 = \mathbf{FP}/\mathsf{poly}$;
(2) $\mathsf{rPLL}_2^{\infty} = \mathsf{PLL}_2 = \mathbf{FP}$.

where, with little abuse of notation, we identify proof systems with the classes of functions they represent. The soundness theorem (i.e., $\mathsf{wrPLL}_2^{\infty} \subseteq \mathbf{FP}/\mathsf{poly}$ and $\mathsf{rPLL}_2^{\infty} \subseteq \mathbf{FP}$) relies on a polynomial modulus of continuity for cut-elimination (Lemma 55), from which we can extract a family of polynomial size circuits computing that functions, which is uniform whenever the coderivation is regular. The completeness theorem (i.e., $\mathbf{FP}/\mathsf{poly} \subseteq \mathsf{wrPLL}_2^{\infty}$ and $\mathbf{FP} \subseteq \mathsf{rPLL}_2^{\infty}$) is established by defining an encoding of a polytime Turing machine with (polynomial) advice in a type system designed to express computation with access to bits of streams (Theorem 75), and by translating the type system into nuPLL_2 (Theorem 81), leveraging on previous translations (Theorem 36).

6.1. **Soundness.** We start with the notions of exponential flow and the notion of rank. Roughly, the rank of a coderivation carries information about the maximum number of nwb calls that can be "extracted" by the cut-elimination step ($\mathsf{c!p}$-vs-$?\mathsf{b}$).

Definition 47 (Exponential flow and rank)**.** Let $\mathcal{D} \in \mathsf{wrPLL}_2^\infty$ The **exponential graph of** $\mathcal{D}$, written $\mathcal{G}(\mathcal{D})$, is a directed acyclic graph whose nodes are the ?-formulas and the !-formulas of $\mathcal{D}$ with nesting level 0, and such that there is an edge from a node A to a node B whenever:

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

An **exponential flow** is a finite directed path in $\mathcal{G}(\mathcal{D})$. Exponential flows range over ϕ. The **rank of an exponential flow** ϕ, written $\mathsf{rk}_\mathcal{D}(\phi)$, is the number of principal formulas of a $?\mathsf{b}$ rule crossed by the path. Given $A \in \mathcal{G}(\mathcal{D})$ an exponential formula such that $\mathbf{nl}_\mathcal{D}(A) = 0$, we denote by $\mathsf{rk}_\mathcal{D}(A) \in \mathbb{N} \cup \{\infty\}$ the supremum of the ranks the exponential flows starting from A Finally, we denote by $\mathsf{rk}(\mathcal{D}) \in \mathbb{N} \cup \{\infty\}$ the supremum of the ranks the exponential flows in $\mathcal{D}$.

Proposition 48. *Let $\mathcal{D} \in \mathsf{wrPLL}_2^\infty$ and A be an exponential formula such that $\mathbf{nl}_\mathcal{D}(A) = 0$. Then $\mathsf{rk}_\mathcal{D}(A) \in \mathbb{N}$ and $\mathsf{rk}(\mathcal{D}) \in \mathbb{N}$.*

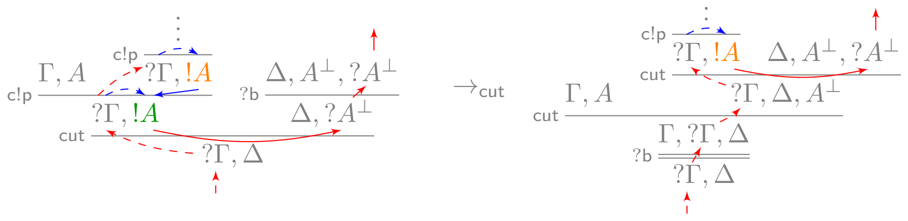
Proof. It is a consequence of the fact that only formulas with nesting level 0 are in $\mathcal{G}(\mathcal{D})$, and byProposition 40 only finitely many formulas of them are principal for a $?\mathsf{b}$ rule. □

Proposition 49. *Let $\mathcal{D} \in \mathsf{wrPLL}_2^\infty$ with conclusion Γ !-free. Then any exponential flow ϕ in $\mathcal{D}$ ends with the principal formula of a $?\mathsf{w}$ rule.*

Proof. It follows by inspecting the inference rules for wrPLL_2^∞, using the fact that instantiations of $\exists$ are $!, ?$-free. □

Proposition 50. *Let $\mathcal{D} \in \mathsf{wrPLL}_2^\infty$ and let $\mathcal{D} \to_{\mathsf{cut}} \mathcal{D}'$. Then, for any occurrence of a !-formula $A \in \mathcal{G}(\mathcal{D})$ if $A \in \mathcal{G}(\mathcal{D}')$ then $\mathsf{rk}_{\mathcal{D}'}(A) \leq \mathsf{rk}_\mathcal{D}(A)$.*

Proof. The only interesting case is when the cut-elimination step is applied to the cut ($\mathsf{c!p}$-vs-$?\mathsf{b}$). By representing bundles of edges with dashed lines, we can illustrate the edges of $\mathcal{G}(\mathcal{D})$ and $\mathcal{G}(\mathcal{D}')$ affected by that cut-elimination step as follows:



where we use colours to distinguish between occurrences of the same formula. Focusing on !-formulas, we notice that $!A$ is in $\mathcal{D}$ but not in $\mathcal{D}'$. On the other hand, $!A$ is in both coderivations and satisfies $\mathsf{rk}_{\mathcal{D}'}(!A) \leq \mathsf{rk}_{\mathcal{D}}(!A)$, as the $?\mathsf{b}$ rule in $\mathcal{D}$ has been replaced by a (possibly empty) series of $?\mathsf{b}$ rules, each one applied to a distinct formula of $?\Gamma$. □

We now define a notion of (finite) size for coderivations in wrPLL_2^∞, relying on **??** and by Remark 26.

Definition 51 (Cosize)**.** Let $\mathcal{D} \in \mathsf{wrPLL}_2^\infty$, and $\mathsf{border}(\mathcal{D}) = \{\nu_1, \dots, \nu_k\}$ be its decomposition, with $\mathfrak{S}_i = \mathcal{D}_{\nu_i}$. We define the **cosize of** $\mathcal{D}$, written $||\mathcal{D}||$, by induction on the depth of $\mathcal{D}$. If $\mathbf{d}(\mathcal{D}) = 0$ then $\mathcal{D} = \mathsf{base}(\mathcal{D})$ and we set $||\mathcal{D}|| := |\mathcal{D}|$. Otherwise $\mathbf{d}(\mathcal{D}) > 0$, and

$$||\mathcal{D}|| := |\mathsf{base}(\mathcal{D})| + \sum_{i\geq 1}^{k} \sum_{\mathcal{D}' \in \{\mathfrak{S}_j(0), \mathfrak{S}_j(1), \dots\}} ||\mathcal{D}'||$$

We also define the **maximal size at depth** d, written $||\mathcal{D}||_d$ with $0 \leq d \leq \mathbf{d}(\mathcal{D})$, by induction on d:

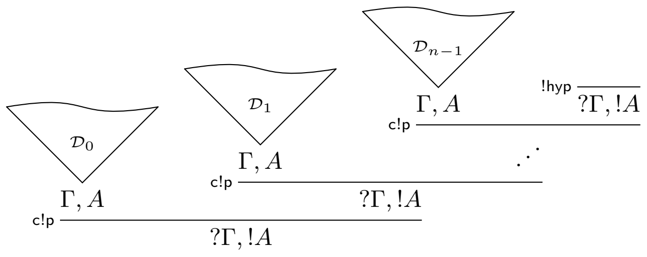
- $||\mathcal{D}||_0 = |\mathsf{base}(\mathcal{D})|$,
- $||\mathcal{D}||_{d+1} = \max_{i=1}^{k} \max_{\mathcal{D}' \in \{\mathfrak{S}_j(0), \mathfrak{S}_j(1), \dots\}} ||\mathcal{D}'||_d$

We now define a particular notion of finite approximation for coderivations called "truncation".

Definition 52 (Finite nwb and truncation)**.** We set

$$\mathsf{!hyp}\,\frac{}{?\Gamma, !A} \quad := \quad \mathsf{c!p}\,\frac{\mathsf{hyp}\,\dfrac{}{\Gamma, A} \qquad \mathsf{hyp}\,\dfrac{}{?\Gamma, !A}}{?\Gamma, !A}$$

A **finite non-wellfounded promotion**, Fnwb, is any coderivation of the form



We denote the above Fnwb with $\mathsf{c!p}_{\langle \mathcal{D}_0, \mathcal{D}_1, \dots, \mathcal{D}_{n-1}\rangle}$. Finite non-wellfounded promotions will range over $\mathfrak{F}$. If $\mathfrak{F} = \mathsf{c!p}_{\langle \mathcal{D}_0, \mathcal{D}_1, \dots, \mathcal{D}_{n-1}\rangle}$ we may write $\mathfrak{F}(i)$ to denote $\mathcal{D}_i$.

Let $\mathcal{D} \in \mathsf{wrPLL}_2^\infty$, let $\mathsf{border}(\mathcal{D}) = \{\nu_1, \dots, \nu_k\}$ be its decomposition, and let $\mathfrak{S}_i = \mathcal{D}_{\nu_i}$. For any $n > 0$ we define the n**-truncation of** $\mathcal{D}$, written $\lfloor \mathcal{D} \rfloor_n$, by induction on $\mathbf{d}(\mathcal{D})$:

- If $\mathbf{d}(\mathcal{D}) = 0$ then we set $\lfloor \mathcal{D} \rfloor_n := \mathcal{D}$.
- If $\mathbf{d}(\mathcal{D}) > 0$ then we set $\lfloor \mathcal{D} \rfloor_n := \mathsf{base}(\mathcal{D})(\mathfrak{F}_1/\nu_1, \dots, \mathfrak{F}_k/\nu_k)$ where, for all $1 \leq i \leq k$, $\mathfrak{F}_i = \mathsf{c!p}_{\langle \lfloor \mathfrak{S}_i(0) \rfloor_n, \lfloor \mathfrak{S}_i(1) \rfloor_n, \dots, \lfloor \mathfrak{S}_i(n-1) \rfloor_n \rangle}$.

Remark 53. Notice that, for $\Gamma = \varnothing$, the rule $\mathsf{!hyp}$ is the **co-weakening rule** ($\mathsf{!w}$) from differential linear logic [40]. Notice that, as opposed to the rule hyp, whose interpretation is the empty set, we have $\left\{\!\!\left\{ \mathsf{!hyp}\,\frac{}{!A} \right\}\!\!\right\} = \{\, (\vec{[\ ]}, [\ ]) \,\}$.

Clearly, Fnwbs are approximations of nwb. Also, as $\lfloor \mathcal{D} \rfloor_n$ is finite, we can relate its size with the cosize of $\mathcal{D}$:

Proposition 54. *Let $\mathcal{D} \in \mathsf{wrPLL}_2^\infty$. Then*
$$||\lfloor \mathcal{D} \rfloor_n| \in \mathcal{O}(n^{\mathbf{d}(\mathcal{D})+1} \cdot ||\mathcal{D}||^{\mathbf{d}(\mathcal{D})+1})$$

Proof. Let $\mathsf{border}(\mathcal{D}) = \{\nu_1, \dots, \nu_k\}$ be the decomposition of $\mathcal{D}$, with $\mathfrak{S}_i = \mathcal{D}_{\nu_i}$. We prove the statement by induction on $\mathbf{d}(\mathcal{D})$. If $\mathbf{d}(\mathcal{D}) = 0$ then $||\lfloor \mathcal{D} \rfloor_n| = |\mathcal{D}| = ||\mathcal{D}||$. If $\mathbf{d}(\mathcal{D}) = d > 0$ then $||\mathcal{D}|| := |\mathsf{base}(\mathcal{D})| + \sum_{i\geq 1}^{k} \sum_{\mathcal{D}' \in \{\mathfrak{S}_j(0), \mathfrak{S}_j(1), \dots\}} ||\mathcal{D}'||$. By induction hypothesis, $||\lfloor \mathfrak{S}_i(j) \rfloor_n| \in \mathcal{O}(n^d \cdot ||\mathfrak{S}_i(j)||^d)$, and so $||\lfloor \mathfrak{S}_i(j) \rfloor_n| \in \mathcal{O}(n^d \cdot ||\mathcal{D}||^d)$. Moreover, $k \leq ||\mathcal{D}||$. This means that $||\lfloor \mathcal{D} \rfloor_n| \in \mathcal{O}(||\mathcal{D}|| + n \cdot ||\mathcal{D}|| \cdot n^d \cdot ||\mathcal{D}||^d)$, so that $||\lfloor \mathcal{D} \rfloor_n| \in \mathcal{O}(n^{d+1} \cdot ||\mathcal{D}||^{d+1})$. $\square$

Lemma 55 (Polynomial modulus of continuity)**.** *Let $\mathcal{D} \in \mathsf{wrPLL}_2^\infty$ 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 in a finite number of steps to a cut-free hyp-free derivation.*

Proof. We define a finite cut-elimination strategy on $\mathcal{D}$ that applies only cut-elimination steps to cut rules with nesting level 0 (hence, it never reduces ($\mathsf{c!p}$-vs-$\mathsf{c!p}$) steps) and we show that, for some polynomial $p : \mathbb{N} \to \mathbb{N}$ depending solely on $\mathbf{d}(\mathcal{D})$, only nodes in $\lfloor \mathcal{D} \rfloor_{p(||\mathcal{D}||)}$ are visited by the cut-elimination steps.

The cut-elimination strategy is divided into $\mathbf{d}(\mathcal{D}) + 1$ rounds: for any $0 \leq d \leq \mathbf{d}(\mathcal{D})$ the d-th round rewrites a coderivation $\mathcal{D}^d$ to a coderivation $\mathcal{D}^{d+1}$, where $\mathcal{D}^0 := \mathcal{D}$ and $\mathcal{D}^{\mathbf{d}(\mathcal{D})} = f(\mathcal{D})$. Let $\mathsf{border}(\mathcal{D}^d) = \{\nu_1^d, \dots, \nu_{k_d}^d\}$ be the decomposition of $\mathcal{D}^d$, with $\mathfrak{S}_i^d = \mathcal{D}^d_{\nu_i^d}$. Each round d is divided in two phases.

- **Phase 1.** Starting from $\mathcal{D}^d$, we eliminate all linear, commutative and exponential cuts with nesting level 0 except those with cut-formula the principal !-formula of a nwb $\mathfrak{S}_i^d$. We obtain a coderivation $\mathcal{D}^{(d)}$.
- **Phase 2.** We apply to $\mathcal{D}^{(d)}$ exponential cut-elimination steps and commuting steps to cut rules with cut-formula the principal !-formula of a nwb $\mathfrak{S}_i^d$.

Let $\mathsf{rk} := \max_{d=0}^{\mathbf{d}(\mathcal{D})} \mathsf{rk}(\mathcal{D}^d) + 1$. First, for all $0 \leq d \leq \mathbf{d}(\mathcal{D})$, we show by induction on $\mathbf{d}(\mathcal{D}^d)$ that $\lfloor \mathcal{D}^d \rfloor_{\mathsf{rk}}$ rewrites to a cut-free and hyp-free derivation:

- if $\mathbf{d}(\mathcal{D}^d) = 0$ then $d = \mathbf{d}(\mathcal{D})$ and $\lfloor \mathcal{D}^d \rfloor_{\mathsf{rk}} = \mathcal{D}^d$ rewrites to a cut-free (and hyp-free) derivation.
- Let us suppose that $\mathbf{d}(\mathcal{D}^d) > 0$. **Phase 1** only affects $\mathsf{base}(\mathcal{D}^d)$. In particular, $\mathbf{d}(\mathcal{D}^d) = \mathbf{d}(\mathcal{D}^{(d)})$. Moreover, since no linear or commuting cuts are left, and Γ is !-free, $\mathcal{D}^{(d)}$ contains exactly k_d cuts with nesting level 0, say $\mathsf{r}_1^d, \dots, \mathsf{r}_{k_d}^d$, where r_i^d has the principal !-formula of $\mathfrak{S}_i^d$ as cut-formula. Let us now consider **Phase 2**. Let $(!A_1, ?A_1^\perp)$, $\dots$, $(!A_n, ?A_n^\perp)$ be the list of pairs of cut-formulas of a cut-chain starting from r_i^d. We notice that:
  - $?A_1^\perp, \dots, ?A_n^\perp$ enumerates (with possible repetitions) ?-formulas in the same exponential flow, sat ϕ_i.

– $!A_1, \ldots, !A_n$ enumerates (with possible repetitions) !-formulas in the same nwb, say $\mathfrak{S}^d_i$.

So, by applying Proposition 49 and Proposition 50 we obtain:

(1) $\mathcal{D}^{(d)}(\mathfrak{F}^d_1/\nu^d_1, \ldots, \mathfrak{F}^d_{k_d}/\nu^d_{k_d}) \to^*_{\mathsf{cut}} \mathcal{D}^{d+1}$, where $\mathfrak{F}^d_i = \mathsf{c!p}_{\langle \mathfrak{S}^d_i(0), \mathfrak{S}^d_i(1), \ldots, \mathfrak{S}^d_i(\mathsf{rk})\rangle}$.

(2) $\mathbf{d}(\mathcal{D}^d) = \mathbf{d}(\mathcal{D}^{(d)}) - 1 = \mathbf{d}(\mathcal{D}^{d+1}) - 1$.

By Item 2 and the induction hypothesis we have $\lfloor \mathcal{D}^{d+1} \rfloor_{\mathsf{rk}} \to^*_{\mathsf{cut}} f(\mathcal{D})$. By Item 1 we have $\lfloor \mathcal{D}^d \rfloor_{\mathsf{rk}} \to^*_{\mathsf{cut}} \lfloor \mathcal{D}^{(d)} \rfloor_{\mathsf{rk}} \to^*_{\mathsf{cut}} \lfloor \mathcal{D}^{d+1} \rfloor_{\mathsf{rk}}$.

We now show that there is some $k > 0$ depending solely on $\mathbf{d}(\mathcal{D})$ such that $||\mathcal{D}^d||_0 \in \mathcal{O}(||\mathcal{D}||^k)$ for any $0 \leq d \leq \mathbf{d}(\mathcal{D})$. Since $\mathsf{rk}(\mathcal{D}^d) \leq |\mathsf{base}(\mathcal{D}^d)| = ||\mathcal{D}^d||_0$ this provides a bound to the maximum number of calls of a nwb that will be required to compute the input, from which we can conclude that $\lfloor \mathcal{D} \rfloor_{||\mathcal{D}||^k}$ rewrites to a cut-free and hyp-free derivation. We actually show by induction on $0 \leq d \leq \mathbf{d}(\mathcal{D})$ something stronger:

$$||\mathcal{D}^d||_0 \in \mathcal{O}\left(\prod_{i=0}^{d} ||\mathcal{D}^0||_i^{3^{d+1-i}\cdot 2^{d-i}}\right)$$

Notice that this would be enough to conclude, as by Lemma 31,we have:

$$||\mathcal{D}^d||_0 \in \mathcal{O}\left(\prod_{i=0}^{d} ||\mathcal{D}^0||_i^{3^{d+1-i}\cdot 2^{d-i}}\right) = \mathcal{O}\left(\prod_{i=0}^{\mathbf{d}(\mathcal{D})} ||\mathcal{D}||_i^{3^{d+1-i}\cdot 2^{d-i}}\right) = \mathcal{O}\left(\mathbf{d}(\mathcal{D}) \cdot ||\mathcal{D}||^{3^{d+1-i}\cdot 2^{d-i}}\right)$$

The case $d = 0$ is trivial. Concerning the inductive step, we first observe that, by Item 1 we have:

$$\begin{array}{rcl}
||\mathcal{D}^{d+1}||_0 & \in & \mathcal{O}\left(||\mathcal{D}^{(d)}||_0 + \sum_{i=0}^{k_d} \sum_{j=0}^{\mathsf{rk}(\mathcal{D}^d)} ||\mathfrak{S}^d_i(j)||_0\right) \\
& = & \mathcal{O}\left(||\mathcal{D}^{(d)}||_0 + \sum_{i=0}^{k_d} + \mathsf{rk}(\mathcal{D}^{(d)}) \cdot ||\mathcal{D}^{(d)}||_1\right) \\
& = & \mathcal{O}\left(||\mathcal{D}^{(d)}||_0 + \sum_{i=0}^{k_d} + ||\mathcal{D}^{(d)}||_0 \cdot ||\mathcal{D}^{(d)}||_1\right) \\
& = & \mathcal{O}\left(||\mathcal{D}^{(d)}||_0 + ||\mathcal{D}^{(d)}||_0^2 \cdot ||\mathcal{D}^{(d)}||_1\right) \\
& = & \mathcal{O}\left(||\mathcal{D}^{(d)}||_0^2 \cdot ||\mathcal{D}^{(d)}||_1\right)
\end{array}$$

since $k_d, \mathsf{rk}(\mathcal{D}^{(d)}) \leq ||\mathcal{D}^{(d)}||_0$ and $||\mathfrak{S}^d_i(j)||_0 \leq ||\mathcal{D}^{(d)}||_1$. Moreover, by Proposition 39 we have $||\mathcal{D}^{(d)}||_0 \in \mathcal{O}(||\mathcal{D}^d||_0^3)$. Finally, we notice that $||\mathcal{D}^{(d)}||_1 = ||\mathcal{D}^d||_1 = ||\mathcal{D}^0||_{d+1}$ as the rules of $\mathcal{D}^0$ with nesting level d are untouched by the first $d-1$ rounds of cut-elimination and each round decreases the nesting level. By putting everything together we have:

$$\begin{array}{rcl}
||\mathcal{D}^{d+1}||_0 & \in & \mathcal{O}\left(||\mathcal{D}^{(d)}||_0^2 \cdot ||\mathcal{D}^{(d)}||_1\right) \\
& = & \mathcal{O}\left(||\mathcal{D}^{(d)}||_0^2 \cdot ||\mathcal{D}^{0}||_{d+1}\right) \\
& = & \mathcal{O}\left(\left(||\mathcal{D}^{d}||_0^2 \cdot ||\mathcal{D}^{0}||_{d+1}\right)^3\right) \\
& = & \mathcal{O}\left(\left(\left(\prod_{i=0}^{d} ||\mathcal{D}^0||_i^{3^{d+1-i}\cdot 2^{d-i}}\right)^2 \cdot ||\mathcal{D}^{0}||_{d+1}\right)^3\right) \\
& = & \mathcal{O}\left(\left(\prod_{i=0}^{d} ||\mathcal{D}^0||_i^{3^{d+1-i}\cdot 2^{d+1-i}} \cdot ||\mathcal{D}^{0}||_{d+1}\right)^3\right) \\
& = & \mathcal{O}\left(\left(\prod_{i=0}^{d} ||\mathcal{D}^0||_i^{3^{d+1-i}\cdot 2^{d+1-i}}\right)^3 \cdot ||\mathcal{D}^{0}||_{d+1}^3\right) \\
& = & \mathcal{O}\left(\prod_{i=0}^{d} ||\mathcal{D}^0||_i^{3^{d+2-i}\cdot 2^{d+1-i}} \cdot ||\mathcal{D}^{0}||_{d+1}^3\right) \\
& = & \mathcal{O}\left(\prod_{i=0}^{d+1} ||\mathcal{D}^0||_i^{3^{d+2-i}\cdot 2^{d+1-i}}\right)
\end{array}$$

This concludes the proof. □

We can now state and prove the soundness theorem.

Theorem 56 (Soundness)**.** *Let* $f : (\{\mathbf{0},\mathbf{1}\}^*)^n \to \{\mathbf{0},\mathbf{1}\}^*$*:*

(1) If f is representable in wrPLL_2^∞ *then* $f \in \mathbf{FP}/\mathsf{poly}$*;*
(2) If f is representable in rPLL_2^∞ *then* $f \in \mathbf{FP}$.

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

$$\mathcal{D}_{f(s)} \quad \coloneqq \quad \mathsf{cut}\,\dfrac{\overset{\underline{s}}{\mathbf{S}[]} \qquad \overset{\underline{f}}{\mathbf{S}[]^{\perp}, \mathbf{S}[]}}{\mathbf{S}[]}$$

By Lemma 55 there are $\mathcal{D}_0, \mathcal{D}_1, \dots, \mathcal{D}_m$ such that:

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

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, $||\mathcal{D}_{f(s)}|| \in \mathcal{O}(||\underline{s}||) = \mathcal{O}(|\underline{s}|) = \mathcal{O}(|s|)$, where $|s|$ is the size of the string s. So, we have:

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

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

- By Proposition 54 we have

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

  for some $h > 0$ depending solely on $\mathbf{d}(\underline{f})$.
- By Proposition 39, we have both $m \in \mathcal{O}(|s|^{3h})$ and $|\mathcal{D}_i| \in \mathcal{O}(|s|^{3h})$.

This means that we can construct a polysize family of circuits $\mathcal{C} = (C_n)_{n\geq 0}$ such that, for any $n \geq 0$, on input $s = b_1, \dots, b_n \in \{\mathbf{0}, \mathbf{1}\}^*$, $C_n(s)$ evaluates $\mathcal{D}_{f(s)}$ to $\underline{f(s)}$ and returns $f(s)$. Therefore, $f \in \mathbf{FP}/\mathsf{poly}$. Suppose now that 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, $f \in \mathbf{FP}$. □

6.2. **Completeness.** In this subsection we introduce the type system nuPTA_2, which is able to express computation over streams of data, and we show that nuPTA_2 is complete for $\mathbf{FP}/\mathsf{poly}$ (Theorem 75.1). By inspecting the proof of completeness, we will infer that the subsystem of nuPTA_2 without typing rules for streams, called PTA_2, is complete for $\mathbf{FP}$ (Theorem 75.2). As a matter of convenience, nuPTA_2 is endowed with a special modality, denoted "ω", to lift data of type σ to streams over those data. The proof of completeness is based on the encoding of Turing machines given in [30, 31].

6.3. **The type assigment systems PTA_2 and nuPTA_2.**

Definition 57. $[\Lambda_{\mathsf{stream}}]$ We will denote with $\Lambda_{\mathsf{stream}}$ the set of terms generated by the following grammar:

$$\begin{aligned} M \coloneqq{}& x \mid \mathsf{I} \mid \mathsf{let}\ \mathsf{I} = x\ \mathsf{in}\ M \mid M \otimes M \mid \mathsf{let}\ x_1 \otimes x_2 = M\ \mathsf{in}\ M \\ & \lambda x.M \mid MM \mid \mathbf{M} \mid \mathsf{discard} \mid \mathsf{pop} \end{aligned}$$

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

$$\begin{aligned} (\lambda x.M)N &\to_\beta M[N/x] \\ \mathsf{let}\ \mathsf{I} = \mathsf{I}\ \mathsf{in}\ M &\to_\beta M \\ \mathsf{let}\ x_1 \otimes x_2 = M \otimes N\ \mathsf{in}\ P &\to_\beta P[M/x_1, N/x_2] \\ \mathsf{pop}\,\mathbf{M} &\to_\beta hd(\mathbf{M}) \otimes tl(\mathbf{M}) \\ \mathsf{discard}\,(M \otimes \mathbf{M}) &\to_\beta M \end{aligned}$$

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

The n-ary tensor product (with $n \geq 3$) can be easily defined from the binary one as follows:

$$\begin{aligned} M_1 \otimes \dots \otimes M_n &\coloneqq (M_1 \otimes \dots \otimes M_{n-1}) \otimes M_n \\ \sigma_1 \otimes \dots \otimes \sigma_n &\coloneqq (\sigma_1 \otimes \dots \otimes \sigma_{n-1}) \otimes \sigma_n \\ \mathsf{let}\ x_1 \otimes \dots \otimes x_n = z\ \mathsf{in}\ M &\coloneqq \mathsf{let}\ y \otimes x_n = z\ \mathsf{in}\ (\mathsf{let}\ x_1 \otimes \dots \otimes x_{n-1} = y\ \mathsf{in}\ M) \end{aligned}$$

We set $\sigma^n \coloneqq \sigma \otimes \overset{n}{\dots} \otimes \sigma$.

The types of PTA_2 and nuPTA_2 are based on the so-called *essential types* from [31].

Definition 58 (PTA_2 and nuPTA_2)**.** The *essential types* are generated by the following grammar:

$$\begin{aligned} A &\coloneqq X \mid \mathbf{1} \mid \sigma \multimap A \mid \forall X.A \\ \sigma &\coloneqq A \mid \sigma \otimes \sigma \mid !\sigma \mid \omega\sigma \end{aligned}$$

$$\mathsf{ax}\,\frac{}{x : A \vdash x : A} \qquad \multimap_i \frac{\Gamma, x : \sigma \vdash M : B}{\Gamma \vdash \lambda x.M : \sigma \multimap B} \qquad \multimap_e \frac{\Gamma \vdash M : \sigma \multimap B \quad \Delta \vdash N : \sigma}{\Gamma, \Delta \vdash MN : B}$$

$$\mathsf{I}_i\,\frac{}{\vdash \mathsf{I} : \mathbf{1}} \qquad \mathsf{I}_e\,\frac{\Gamma \vdash N : \mathbf{1} \quad \Delta \vdash M : \sigma}{\Gamma, \Delta \vdash \mathsf{let\ I} = N \mathsf{\ in\ } M : \sigma}$$

$$\otimes_i \frac{\Gamma \vdash M : \sigma \quad \Delta \vdash N : \tau}{\Gamma, \Delta \vdash M \otimes N : \sigma \otimes \tau} \qquad \otimes_e \frac{\Gamma \vdash M \otimes N : \sigma \otimes \tau \quad \Delta, x : \sigma, y : \tau \vdash P : C}{\Gamma, \Delta \vdash \mathsf{let\ } x \otimes y = M \otimes N \mathsf{\ in\ } P : C}$$

$$\forall_i \frac{\Gamma \vdash M : A}{\Gamma \vdash M : \forall X.A} \qquad \forall_e \frac{\Gamma \vdash M : \forall X.A}{\Gamma \vdash M : A[B/X]}\ B \text{ is } (!, \omega)\text{-free}$$

$$\mathsf{f!p}\,\frac{\Gamma \vdash M : \sigma}{!\Gamma \vdash M : !\sigma} \qquad ?\mathsf{w}\,\frac{\Gamma \vdash M : \tau}{\Gamma, x : !\sigma \vdash M : \tau} \qquad ?\mathsf{b}\,\frac{\Gamma, y : \sigma, z : !\sigma \vdash M : \tau}{\Gamma, x : !\sigma \vdash M[x/y, x/z] : \tau}$$

$$\mathsf{stream}\,\frac{\vdash \mathbf{M} : (0) : \sigma \quad \vdash \mathbf{M} : (1) : \sigma \quad \dots \quad \vdash \mathbf{M} : (n) : \sigma \quad \dots}{\vdash \mathbf{M} : \omega\sigma}\,(\star)$$

$$\mathsf{discard}\,\frac{}{\vdash \mathsf{discard} : \sigma \otimes \omega\sigma \multimap \sigma} \qquad \mathsf{pop}\,\frac{}{\vdash \mathsf{pop} : \omega\sigma \multimap \sigma \otimes \omega\sigma}$$

FIGURE 16. Typing rules for system nuPTA_2.

where X ranges over a countable set of type variables. We denote with $\sigma[\tau/X]$ the standard meta level substitution of τ for the free occurrences of the type variable X in σ. A *context* is a set with shape $x_1 : \sigma_1, \dots, x_n : \sigma_n$ for some $n \geq 0$, where x_i are term variables and σ_i are types. Context range over $\Gamma, \Delta, \Sigma, \dots$. With $!\Gamma$ we denote a context of the form $x_1 : !\sigma_1, \dots, x_n : !\sigma_n$. The type assignment system for $\Lambda_{\mathsf{stream}}$, called nuPTA_2, derives *judgements* of the form $\Gamma \vdash M : \sigma$ and is given by the typing rules in Figure 16 with the following condition on the rule stream:

$$(\star) \quad \sharp\{\mathbf{M}(0), \mathbf{M}(1), \dots\} \in \mathbb{N}$$

the restriction of nuPTA_2 without the typing rules stream, $\mathsf{discard}$ and pop is called PTA_2. We write $\Gamma \vdash_{\mathsf{nuPTA}_2} M : \sigma$ (resp. $\Gamma \vdash_{\mathsf{PTA}_2} M : \sigma$) when the judgement $\Gamma \vdash M : \sigma$ is derivable in nuPTA_2 (resp. PTA_2). We simply write $\Gamma \vdash M : \sigma$ when the judgement is derivable in both systems. If $\mathcal{D}$ is a typing derivation of $\Gamma \vdash M : \sigma$ then we write $\mathcal{D} : \Gamma \vdash M : \sigma$.

Proposition 59. *If $\mathcal{D} : \Gamma \vdash M : !\sigma$ then $\Gamma = !\Gamma'$ and $\mathcal{D}$ is obtained from $\mathcal{D}'$ by applying* $\mathsf{f!p}$ *followed by a series of applications of* $?\mathsf{w}$ *and* $?\mathsf{b}$.

Lemma 60 (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 the the typing derivation $\mathcal{D}_1$ and the complexity of τ. We only consider the case where $\mathcal{D}_1$ is obtained from $\mathcal{D}'_1$ by applying a $?\mathsf{b}$ rule. This means that $M = M'[x/y, x/z]$ and $\tau = !\tau'$. So, $\mathcal{D}'_1$ has shape:

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

By Proposition 59 we have that $\Delta = !\Sigma$ and

$$\mathcal{D}_2 := {\scriptstyle ?\mathsf{b},?\mathsf{w}}\ \begin{array}{c} \mathcal{D}'_2 \\ !\Sigma' \vdash N' : !\tau' \\ \hline\hline !\Sigma \vdash N : !\tau' \end{array} \qquad\qquad \mathcal{D}'_2 := {\scriptstyle \mathsf{f!p}}\ \begin{array}{c} \mathcal{D}''_2 \\ \Sigma' \vdash N' : \tau' \\ \hline !\Sigma' \vdash N' : !\tau' \end{array}$$

By induction hypothesis on $\mathcal{D}'_1$ and $\mathcal{D}'_2$ we obtain a typing derivation $S(\mathcal{D}'_1, \mathcal{D}'_2)$ of $\Gamma, !\Sigma' \vdash M'[N'/z]\rangle : \sigma$. By applying the induction hypothesis again, we have 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 ?b rules and ?w rules. □

Proposition 61 (Subject reduction)**.** *Let $\mathcal{D} : \Gamma \vdash M : A$. If $M \to_\beta M'$ then there is $\mathcal{D}'$ such that $\mathcal{D}' : \Gamma \vdash M' : A$.*

Proof. It suffices to check that the reduction rules given in Definition 57 preserve types. We only consider the case $M = (\lambda x.P)N$ and $M' = P[N/x]$. The proof is by induction on $\mathcal{D}$. We only show the case where the last rule of $\mathcal{D}'$ is $\multimap_e$, i.e. we have

$$\multimap_e\ \begin{array}{cc} \mathcal{D}_1 & \mathcal{D}_2 \\ \Gamma \vdash \lambda x.P : \sigma \multimap B & \Delta \vdash N : \sigma \\ \hline \multicolumn{2}{c}{\Gamma, \Delta \vdash (\lambda x.P)N : B} \end{array}$$

By inspecting the typing rules, it is easy to check that $\mathcal{D}_1$ has the following shape:

$$\begin{array}{rc} & \mathcal{D}'_1 \\ & \Gamma', x : \sigma \vdash P' : B \\ \multimap_i & \\ \hline & \Gamma' \vdash \lambda x.P' : \sigma \multimap B \\ {?\mathsf{b},?\mathsf{w}} & \\ \hline\hline & \Gamma \vdash \lambda x.P : \sigma \multimap B \end{array}$$

We apply Lemma 60 and we obtain the following derivation:

$${\scriptstyle ?\mathsf{b},?\mathsf{w}}\ \begin{array}{c} S(\mathcal{D}'_1, \mathcal{D}_2) \\ \Gamma', \Delta \vdash P'[N/x] : B \\ \hline\hline \Gamma, \Delta \vdash P[N/x] : B \end{array}$$

□

6.4. **Definability and basic data types in PTA_2 and nuPTA_2.** We generalise the usual notion of lambda definability given in [41] to different kinds of input data, along the lines of [31].

Definition 62 (Representability)**.** 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 $0 \le j \le n$ 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 $\underline{f} \in \Lambda_{\mathsf{stream}}$ such that $\vdash \underline{f} : \mathbf{I}_1 \to \ldots \to \mathbf{I}_n \to \mathbf{O}$ in nuPTA_2 (resp. 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 $M^0(N) \coloneqq N$ and $M^{n+1}(N) \coloneqq M(M^n(N))$. We also set $M \circ N \coloneqq \lambda z.M(Nz)$, which generalises to the n-ary case $M_1 \circ \ldots \circ M_n \coloneqq \lambda z.M_1(M_2(\ldots(M_n z)))$.

In what follows we show encodings and properties of basic data types (such as booleans, boolean strings and natural numbers) and operations on those data types (such as iterations and polynomials).

Definition 63 (Booleans)**.** We define the type of Booleans as $\mathbf{B} \coloneqq \forall X.(X \otimes X) \multimap X \otimes X$. We also define the boolean values and basic operations on them as follows:

$$\begin{array}{rcll}
\underline{\mathbf{1}} & \coloneqq & \lambda x.\lambda y.x \otimes y & : \mathbf{B} \\
\underline{\mathbf{0}} & \coloneqq & \lambda x.\lambda y.y \otimes x & : \mathbf{B} \\
\mathbf{W_B} & \coloneqq & \lambda b.\mathsf{let}\ x_1 \otimes x_2 = b(\mathsf{I} \otimes \mathsf{I})\ \mathsf{in}\ \mathsf{let}\ \mathsf{I} = x_2\ \mathsf{in}\ x_1 & : \mathbf{B} \multimap \mathbf{1} \\
\pi^2_1 & \coloneqq & \lambda x.\mathsf{let}\ x_1 \otimes x_2 = x\ \mathsf{in}\ \mathsf{let}\ \mathsf{I} = \mathbf{W_B}\, x_2\ \mathsf{in}\ x_1 & : \mathbf{B} \otimes \mathbf{B} \multimap \mathbf{B} \\
\mathbf{C_B} & \coloneqq & \lambda b.\pi^2_1(b(\underline{\mathbf{0}} \otimes \underline{\mathbf{0}}) \otimes (\underline{\mathbf{1}} \otimes \underline{\mathbf{1}})) & : \mathbf{B} \multimap \mathbf{B} \otimes \mathbf{B} \\
\underline{\neg} & \coloneqq & \lambda b.\lambda x.\lambda y.b(y \otimes x) & : \mathbf{B} \multimap \mathbf{B} \\
\underline{\vee} & \coloneqq & \lambda b_1.\lambda b_2.\pi^2_1(b_1 \underline{\mathbf{0}}\, b_2) & : \mathbf{B} \otimes \mathbf{B} \multimap \mathbf{B}
\end{array}$$

A consequence of the above encoding is the following:

Proposition 64 (Functional completeness)**.** *Each boolean function $f : \mathbb{B}^n \to \mathbb{B}^m$ with $n \geq 0$, $m > 0$ can be defined by a term $\underline{f} \in \Lambda_{\mathsf{stream}}$ such that $\vdash \underline{f} : \mathbf{B}^n \multimap \mathbf{B}^m$.*

Definition 65 (Π_1 and $e\Pi_1$ types [30])**.** Let A be a type build from $\mathbf{1}, \otimes, \multimap, \forall$. We say that A is Π_1 if $\forall$-types occur only positively in it. We say that A is $e\Pi_1$ if it contains positive occurrences of $\forall$-types or negative occurrences of inhabited $\forall$-types.

Proposition 66 (Linear weakening [30])**.** *For any closed type $A \in e\Pi_1$ there is a term $\mathbf{W}_A$ in the linear λ-calculus that inhabits $A \multimap \mathbf{1}$.*

Remark 67 (Conditional)**.** The above proposition allows us to derive the following conditional rule for any closed type $A \in e\Pi_1$:

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

where $\mathsf{if}\ x\ \mathsf{then}\ R\ \mathsf{else}\ L \coloneqq \pi^2_1(xRL)$. It is easy to check that the following reduction hold:

$$\begin{array}{lcl}
\mathsf{if}\ \underline{\mathbf{1}}\ \mathsf{then}\ R\ \mathsf{else}\ L & \to^*_\beta & R \\
\mathsf{if}\ \underline{\mathbf{0}}\ \mathsf{then}\ R\ \mathsf{else}\ L & \to^*_\beta & L
\end{array}$$

Definition 68 (Natural numbers and boolean strings)**.** Natural numbers and Boolean string are encoded by the following types:

$$\begin{array}{rcl}
\mathbf{N} & \coloneqq & \forall X.!(X \multimap X) \multimap X \multimap X \\
\mathbf{S} & \coloneqq & \forall X.!(\mathbf{B} \multimap X \multimap X) \multimap X \multimap X
\end{array}$$

We will mainly use a parametric version of the above types. For any linear type A we set:

$$\begin{array}{rcl}
\mathbf{N}[A] & \coloneqq & !(A \multimap A) \multimap A \multimap A \\
\mathbf{S}[A] & \coloneqq & !(\mathbf{B} \multimap A \multimap A) \multimap A \multimap A
\end{array}$$

We write $\mathbf{N}[]$ to denote $\mathbf{N}[A]$ for some A, and similarly for $\mathbf{S}$. The encoding of boolean strings and natural numbers are defined as follows, for $s = b_1 \cdots b_n \in \{0,1\}^*$:

$$\begin{aligned} \underline{s} &:= \lambda f.\lambda z.f\, \underline{s_n}(f\, \underline{s_{n-1}}(\ldots(f\, \underline{s_1}\, z)\ldots)) \\ \underline{n} &:= \lambda f.\lambda z.f^n z \end{aligned}$$

We define the term length that, when applied to the encoding of a string, returns the encoding of its length:

$$\mathsf{length} := \lambda s.\lambda f.s(\lambda x.\lambda y.\mathsf{let}\ \mathsf{I} =\ \mathbf{W_B}\, x\ \mathsf{in}\ (f)y) \quad : \mathbf{S}[A] \multimap \mathbf{N}[A]$$

It is easy to check that, if $s = b_1 \cdots b_n$ then

$$\mathsf{length}\, \underline{s} \to^*_\beta \underline{n}$$

Remark 69. In type systems based on second-order linear logic, functions over boolean strings and natural numbers are typically encoded by terms of type $\mathbf{S} \multimap \mathbf{S}$ and $\mathbf{N} \multimap \mathbf{N}$, where $\mathbf{S}$ and $\mathbf{N}$ are as in Definition 68. However, these types are too restrictive for establishing our completeness theorem (Theorem 75). Indeed, to represent a Turing machine we need to iterate (the encoding of) a transition function $\mathsf{Tr} : \mathbf{TM} \multimap \mathbf{TM}$ (where $\mathbf{TM}$ is the type of Turing machine configurations) as many times as a given natural number n. This can be done by applying $\underline{n}$ to Tr, which can be typed using the rule $\forall_e$ turning $\mathbf{N}$ into its instantiation $\mathbf{N}[\mathbf{TM}]$. However, $\mathbf{TM}$ makes crucial use of ! modalities, so that we cannot legally apply $\forall_e$. Following [27], we overcome this issue by switching to the types $\mathbf{S}[]$ and $\mathbf{N}[]$, so that the term $\underline{n}\,\mathsf{Tr}$ is typable as long as $\underline{n}$ has type $\mathbf{N}[\mathbf{TM}]$.

Definition 70 (Iteration)**.** We derive the iterator on natural numbers with result type A as follows:

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

where $\mathsf{iter}_\mathbf{N} := \lambda\, n.\lambda s.\lambda b.nsb$. It is easy to check that the reduction below holds:

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

Similarly, we define the iterator on boolean strings with result type A:

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

where $\mathsf{iter}_\mathbf{S} := \lambda\, s\lambda t.\lambda u.\lambda b.stub$. It is easy to check that the reduction below holds:

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

Remark 71 (Tiering)**.** Let σ be a type. We define $\mathbf{N}_\sigma[d]$ and $\mathbf{S}_\sigma[d]$ by induction on $d \geq 0$:

$$\begin{aligned} \mathbf{N}_\sigma[0] &:= \sigma & \mathbf{S}_\sigma[0] &:= \sigma \\ \mathbf{N}_\sigma[d+1] &:= \mathbf{N}_\sigma[\mathbf{N}_\sigma[d]] & \mathbf{S}_\sigma[d+1] &:= \mathbf{S}_\sigma[\mathbf{S}_\sigma[d]] \end{aligned}$$

When σ is clear from the context, we simply write $\mathbf{N}[d]$ and $\mathbf{S}[d]$. We set:

$$\begin{aligned} \mathsf{down}^d_\mathbf{N} &:= \lambda x.\mathsf{iter}_\mathbf{N}\, x\, \mathsf{succ}\, \underline{0} : \mathbf{N}[d+1] \multimap \mathbf{N}[d] \\ \mathsf{down}^d_\mathbf{S} &:= \lambda x.\mathsf{iter}_\mathbf{N}\, x\, (\, \lambda b.\lambda s.\lambda c.\lambda z.cb(scz))\underline{\epsilon} : \mathbf{N}[d+1] \multimap \mathbf{N}[d] \end{aligned}$$

Notice that:

$$\begin{aligned} \mathsf{down}^d_\mathbf{N}\, \underline{n} &\to^*_\beta \underline{n} \\ \mathsf{down}^d_\mathbf{S}\, \underline{n} &\to^*_\beta \underline{n} \end{aligned}$$

Definition 72 (Successor, addition, multiplication)**.** Successor, addition and multiplication can be represented as follows:

$$\begin{array}{rcll} \mathsf{succ} & := & \lambda n.\lambda f.\lambda z.n(f)(fz) & : \mathbf{N}[i] \multimap \mathbf{N}[i] \\ \mathsf{add} & := & \lambda n.\lambda m.\mathsf{iter}_{\mathbf{N}}\, n\,(\mathsf{succ})\, m & : \mathbf{N}[i+1] \multimap \mathbf{N}[i] \multimap \mathbf{N}[i] \\ \mathsf{mult} & := & \lambda n.\lambda m.\mathsf{iter}_{\mathbf{N}}\, m\,(\lambda y.\mathsf{add}\, n\, y)\, \underline{0} & : !\mathbf{N}[i+1] \multimap \mathbf{N}[i+1] \multimap \mathbf{N}[i] \end{array}$$

Theorem 73 (Polynomial completeness)**.** *Let $p(x) : \mathbb{N} \to \mathbb{N}$ be a polynomial, and let $\mathsf{deg}(p)$ be its degree. Then there is a term $\underline{p}$ representing p typable, for any i, as*

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

Proof. W.l.o.g. we will prove the statement for $i = 0$. Consider a polynomial $p(x) : \mathbb{N} \to \mathbb{N}$ in Horner normal form, i.e., $p(x) = a_0 + x(a_1 + x(\ldots(a_{n-1} + x a_n)\ldots))$. We show by induction on $\mathsf{deg}(p)$ something stronger, i.e., for $i > 0$ it is derivable:

$$x_0 : \mathbf{N}[1], x_1 : !\mathbf{N}[2], \ldots, x_{\mathsf{deg}(p^*)-1} : !^{\mathsf{deg}(p^*)-1}\mathbf{N}[\mathsf{deg}(p^*)] \vdash \underline{p^*} : \mathbf{N} \tag{5}$$

where $p^* = a_0 + x_0(a_1 + x_1(\ldots(a_{\deg p^*-2} + x_{\mathsf{deg}(p^*)-1} a_{\mathsf{deg}(p^*)-1})\ldots))$. If $\mathsf{deg}(p^*) = 1$ then $p^* = a_0 + x_0 a_1$, and we simply set $\underline{p^*} := \mathsf{add}\,\underline{a_0}\,(\mathsf{mult}\,\underline{a_1}\,x_0)$. If $\mathsf{deg}(p^*) > 0$ then $p^* = a_0 + x_0 q^*$ with $q^* := a_1 + x_1(a_2 + x_2(\ldots(a_{\mathsf{deg}(p^*)-2} + x_{\mathsf{deg}(p^*)-1} a_{\mathsf{deg}(p^*)-1})\ldots))$. By induction hypothesis we have that q^* is represented by some $\underline{q^*}$ typable as:

$$x_1 : \mathbf{N}[1], x_2 : !\mathbf{N}[2], \ldots, x_{\mathsf{deg}(p^*)-1} : !^{\mathsf{deg}(q^*)-1}\mathbf{N}[\mathsf{deg}(q^*)] \vdash \underline{q^*} : \mathbf{N}$$

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

$$x_1 : \mathbf{N}[2], x_2 : !\mathbf{N}[3], \ldots, x_{\mathsf{deg}(p^*)-1} : !^{\mathsf{deg}(q^*)-1}\mathbf{N}[\mathsf{deg}(q^*)+1] \vdash M : \mathbf{N}$$

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

$$x_0 : \mathbf{N}[1], x_1 : !\mathbf{N}[2], x_2 : !^2\mathbf{N}[3], \ldots, x_{\mathsf{deg}(p^*)-1} : !^{\mathsf{deg}(q^*)}\mathbf{N}[\mathsf{deg}(q^*)+1] \vdash \underline{p^*} : \mathbf{N}$$

and we can conclude since $\mathsf{deg}(q^*) = \mathsf{deg}(p^*) - 1$.

Now, to prove the theorem it suffices to repeatedly apply $\mathsf{down}^k_{\mathbf{N}}$ for appropriate k to the typable term in Equation (5) in order to get a term N that represents p^* and typable as

$$x_0 : \mathbf{N}[\mathsf{deg}(p^*)], x_1 : !\mathbf{N}[\mathsf{deg}(p^*)], \ldots, x_{\mathsf{deg}(p^*)-1} : !^{\mathsf{deg}(p^*)-1}\mathbf{N}[\mathsf{deg}(p^*)] \vdash \underline{p^*} : \mathbf{N}$$

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

Definition 74 (Streams of booleans)**.** The type of streams of booleans is $\mathbf{Stream} := \omega\mathbf{B}$. 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.

6.5. **Encoding polytime Turing machines with polynomial advice in nuPTA_2.** We define the configuration of a Turing machine (with advice) by terms of the form:

$$\lambda c.(cb^l_0 \circ \ldots \circ cb^l_n) \otimes (cb^r_0 \circ \ldots \circ cb^l_m) \otimes Q \otimes \underline{\alpha} \tag{6}$$

where $cb^l_0 \circ \ldots \circ cb^l_n$ and $cb^r_0 \circ \ldots \circ cb^r_m$ represent respectively the left and th right part of the tape, and $Q := b_1 \otimes \ldots \otimes b_k$ represents a tuple of booleans encoding the current state of the machine. We assume w.l.o.g. that the left part of the tape is represented in a reverse order, that the alphabet is composed by the two symbols $\mathbf{1}$ and $\mathbf{0}$, that the scanned symbol is b^r_0 in the right part, and that final states are divided into accepting and rejecting.

Terms as in Equation (6) have the following type:

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

The initial configuration of a Turing machine is represented by a tape of fixed length filled by $\mathbf{0}$s with the head at the beginning of the tape and in the initial state Q_0.

The initial configuration of a Turing machine can be obtained starting from a numeral representing the length of the tape. In particular, it takes a numeral $\underline{n}$ and gives as output a Turing machine with tape of length n filled by $\mathbf{0}$s in the initial state Q_0 and with the head at the beginning of the tape.

$$\mathsf{init} \coloneqq \lambda n.\lambda c.(\lambda z.z) \otimes (\lambda z.n(c\mathbf{0})z) \otimes \underline{Q_0} \otimes \underline{\alpha} : \mathbf{N}[d] \multimap \mathbf{Stream} \multimap \mathbf{TM}$$

for any $d \geq 0$. Following [30, 31], in order to show that Turing machine transitions are definable we consider two distinct phases. In the first one the Turing configuration is decomposed to extract the first symbol of each part of the tape. In the second phase the symbols obtained in the previous one are combined, depending on the transition function, to reconstruct the tape after the transition step. In order to type the decomposition of a Turing machine configuration we will use the type $\mathbf{ID}$ defined as:

$$\mathbf{ID} \coloneqq \forall X.!(\mathbf{B} \multimap X \multimap X) \multimap ((X \multimap X)^2 \otimes (\mathbf{B} \multimap X \multimap X) \otimes \mathbf{B} \otimes (\mathbf{B} \multimap X \multimap X) \otimes \mathbf{B}^k \otimes \mathbf{Stream})$$

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

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

where $F[x] \coloneqq \lambda b.\lambda z.\mathsf{let}\ g \otimes h \otimes i = z\ \mathsf{in}\ (h\, i \circ g) \otimes x \otimes b$, which is typable as:

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

Notice that in dec the variable m has type $\mathbf{TM}$ and is applied to the term $F[c]$. This requires to apply to the variable m the rule $\forall_e$, instantiating the type variable X with the !-free type $((X \multimap X) \otimes (\mathbf{B} \multimap X \multimap X) \otimes \mathbf{B})$.

The term dec in Equation (7) satisfies the following reduction:

$$\begin{aligned}&\mathsf{dec}(\lambda c.(cb_0^l \circ \ldots \circ cb_n^l) \otimes (cb_0^r \circ \ldots \circ cb_m^l) \otimes Q \otimes \underline{\alpha})\\ &\qquad\to_\beta^*\\ &\qquad\lambda c.(cb_1^l \circ \ldots \circ cb_n^l) \otimes (cb_1^r \circ \ldots \circ cb_m^l) \otimes c \otimes b_0^l \otimes c \otimes b_0^r \otimes Q \otimes \underline{\alpha}\end{aligned}$$

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

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

where:

$$\begin{aligned}R &\coloneqq \lambda b'.\lambda q'.\lambda s.\mathsf{let}\ l \otimes r \otimes c_l \otimes b_l \otimes c_r = s\ \mathsf{in}\ (c_r b' \circ c_l b_l \circ l) \otimes r \otimes q'\\ L &\coloneqq \lambda b'.\lambda q'.\lambda s.\mathsf{let}\ l \otimes r \otimes c_l \otimes b_l \otimes c_r = s\ \mathsf{in}\ l \otimes (c_l b_l \circ c_r b' \circ r) \otimes q'\end{aligned}$$

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

The term comp in Equation (8) satisfies the following reduction:

$$\begin{aligned}\mathsf{comp}(\lambda c.(cb^l_1 \circ \ldots \circ cb^l_n) \otimes (cb^r_1 \circ \ldots \circ cb^l_m) \otimes c \otimes b^l_0 \otimes c \otimes b^r_0 \otimes Q \otimes \underline{\alpha})&\\ \to^*_\beta&\\ \lambda c.(cb' \circ cb^l_0 \circ \ldots \circ cb^l_n) \otimes (cb^r_1 \circ \ldots \circ cb^l_m) \otimes Q' \otimes \underline{tl(\alpha)}&\end{aligned}$$

if $\delta(b^r_0, hd(\alpha), Q) = (b', Q', \text{Right})$, and the following reduction:

$$\begin{aligned}\mathsf{comp}(\lambda c.(cb^l_1 \circ \ldots \circ cb^l_n) \otimes (cb^r_1 \circ \ldots \circ cb^l_m) \otimes c \otimes b^l_0 \otimes c \otimes b^r_0 \otimes Q \otimes \underline{\alpha})&\\ \to^*_\beta&\\ \lambda c.(cb^l_1 \circ \ldots \circ cb^l_n) \otimes (cb^l_0 \circ cb' \circ cb^r_1 \circ \ldots \circ cb^l_m) \otimes Q' \otimes \underline{tl(\alpha)}&\end{aligned}$$

if $\delta(b^r_0, hd(\alpha), Q) = (b', Q', \text{Left})$, where δ 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.

By combining the above terms we obtain the encoding of the Turing machine transition step, $\mathsf{Tr} := \mathsf{comp} \circ \mathsf{dec}$, with type $\mathbf{TM} \multimap \mathbf{TM}$.

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

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

where:

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

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

The term In defines a function that, when supplied by a boolean string and a Turing machine, writes the input string on the tape of the Turing machine,

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:

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

where $\mathbf{W}_{\mathbf{B}^{k+1}}$ is the eraser for $\mathbf{B}^{k+1}$ given by Proposition 66.

We can now prove our fundamental theorem:

Theorem 75 (Completeness). *Let* $f : (\{\mathbf{0}, \mathbf{1}\}^*)^n \to \{\mathbf{0}, \mathbf{1}\}^*$*:*

(1) If $f \in \mathbf{FP}/\mathsf{poly}$ *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 unary for the sake of simplicity. Let us prove point Item 1. Let $f \in \mathbf{FP}/\mathsf{poly}$. By Proposition 6 we have $f \in \mathbf{FP}(\mathbb{R})$, so there is a polynomial Turing machine that can make polynomially many queries to bits of a boolean stream α. Let $p(x)$ and $q(x)$ be polynomials bounding the time and space of the Turing machine respectively, with $\mathsf{deg}(p) = m$ and $\mathsf{deg}(q) = l$. By Theorem 73 we obtain $\underline{p}$ and $\underline{q}$ typable as:

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

where $\mathbf{N}[1] = \mathbf{N}[\mathbf{TM}]$, and hence:

$$\begin{array}{c} s' : !^{m-1}\mathbf{S}[m+1] \vdash \underline{p}[\mathsf{length}\, s'/y] : \mathbf{N}[1] \\ s'' : !^{l-1}\mathbf{S}[l+1] \vdash \underline{q}[\mathsf{length}\, s''/z] : \mathbf{N}[1] \end{array}$$

On the other hand, we have:

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

Let $M' := \mathsf{Ext}(((\underline{p}[\mathsf{length}\, s'/y])\,\mathsf{Tr})(\mathsf{In}\, t\, (\mathsf{init}\, (\underline{q}[\mathsf{length}\, s''/z])\, \alpha)))$. By putting everything together we have:

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

By repeatedly applying $?\mathsf{b}$ and $\mathsf{down}^k_{\mathbf{S}}$ for appropriate k we obtain a term M representing f such that:

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

By applying $\mathsf{down}^1_{\mathbf{S}}$ we obtain $x : \mathbf{S}[!^{\max(m,l)}\mathbf{S}[\max(m,l)+1]] \vdash M[\mathsf{down}^1_{\mathbf{S}}\, x/s] : \mathbf{S}$. We set $\underline{f} := M[\mathsf{down}^1_{\mathbf{S}}\, x/s]$, so that $x : \mathbf{S}[] \vdash \underline{f} : \mathbf{S}$.

Point Item 2 is obtained by stripping away the type of streams from the above encoding. $\square$

The following remarks justify the finiteness condition $(\star)$ in nuPTA_2 and the restriction of second order instantiation to $(!, \omega)$-free types.

Remark 76. If the condition $(\star)$ 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 $\omega!\mathbf{N}$ and encoding all values of the function f. We set $A := \mathbf{N}[\mathbf{1}] \otimes \omega\mathbf{N}[\mathbf{1}]$ and define:

$$\begin{array}{rcll} \mathsf{step} & := & \lambda x.\mathsf{let}\ y_1 \otimes y_2 = x\ \mathsf{in}\ \mathsf{let}\ \mathsf{I} =\ y_1\, (\lambda z.z)\, \mathsf{I}\ \mathsf{in}\ \mathsf{pop}\, y_2 & : A \multimap A \\ \underline{f} & := & \lambda n.\mathsf{discard}\, (n\, \mathsf{step}\, (\mathsf{pop}\ \mathbf{F})) & : \mathbf{N}[A] \multimap \mathbf{N}[\mathbf{1}] \end{array}$$

It is easy to check that, for any $n \in \mathbb{N}$, $\underline{f} \to^*_\beta \underline{f(n)}$.

The above observation can be extended to the proof systems nuPLL_2 without the condition $(\star)$ (and, hence, to the proof system pPLL^∞_2) by adapting the above proof.

Remark 77. If the $(!, \omega)$-freeness condition on $\forall_e$ were dropped then nuPTA_2 could represent exponential functions. Indeed, we can define the following functions:

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

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

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

6.6. **Translating type systems into proof systems.**

Definition 78 (Translation)**.** We define a translation $(\_)^\dagger : \mathsf{nuPTA}_2 \to \mathsf{nuPLL}_2$ mapping typing derivations of nuPTA_2 to derivations of nuPLL_2 (and, in particular, typing derivations of PTA_2 to derivations of PLL_2) as follows:

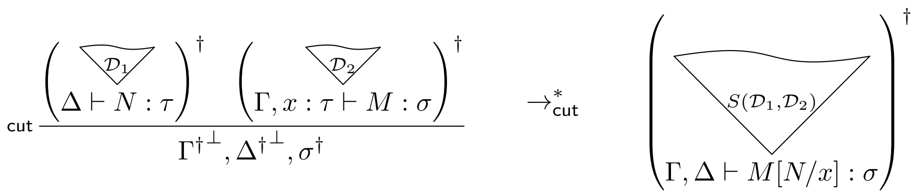
- It maps types of nuPTA_2 to formulas of nuPLL_2:

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

  we notice that $\sigma^\dagger[\tau^\dagger/X] = (\sigma[\tau/X])^\dagger$.
- It maps judgements $\Gamma \vdash M : \tau$ to sequents ${\Gamma^\dagger}^\perp, \tau^\dagger$, where if $\Gamma = x_1 : \sigma_1, \dots, x_n : \sigma_n$ then $\Gamma^\dagger = \sigma_1^\dagger, \dots, \sigma_n^\dagger$.
- It maps a typing rule to gadgets as in Section 6.6 and Section 6.6.

The following two lemmas represent stronger versions of Lemma 60 and Proposition 61 respectively.

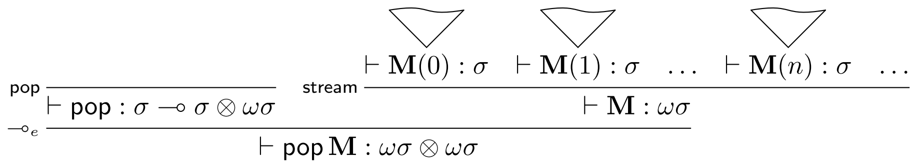
Lemma 79. *For any $\mathcal{D}_1 : \Gamma \vdash M : \sigma$ and $\mathcal{D}_2 : \Delta \vdash N : \tau$ there is $S(\mathcal{D}_1, \mathcal{D}_2)$ such that:*


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

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

Lemma 80. *Let $\mathcal{D}_1 : \Gamma \vdash M_1 : \sigma$ in nuPTA_2. If $M_1 \to_\beta M_2$ then there exist a typing derivation $\mathcal{D}_2 : \Gamma \vdash M_2 : \sigma$ such that $\mathcal{D}_1^\dagger \to^*_{\mathsf{cut}} \mathcal{D}_2^\dagger$.*

Proof. It suffices to check the statement for the reduction rules in Definition 57, by inspecting the cut-elimination rules in Figure 9. We consider the most important cases. If $M_1 = \mathsf{pop}\,\mathbf{M}$ and $M_2 = hd(\mathbf{M}) \otimes tl(\mathbf{M})$, then w.l.o.g. $\mathcal{D}_1$ has the following shape:


pop
$\vdash \mathsf{pop} : \sigma \multimap \sigma \otimes \omega\sigma$
stream
$\vdash \mathbf{M}(0) : \sigma$
$\vdash \mathbf{M}(1) : \sigma$
$\dots$
$\vdash \mathbf{M}(n) : \sigma$
$\dots$
$\vdash \mathbf{M} : \omega\sigma$
$\multimap_e$
$\vdash \mathsf{pop}\,\mathbf{M} : \omega\sigma \otimes \omega\sigma$

$$\mathsf{ax}\,\frac{}{x : A \vdash x : A} \qquad \mapsto \qquad \mathsf{ax}\,\frac{}{A^{\perp}, A}$$

$$\multimap_i\,\frac{\Gamma, x : \sigma \vdash M : B}{\Gamma \vdash \lambda x.M : \sigma \multimap B} \qquad \mapsto \qquad \parr\,\frac{{\Gamma^{\dagger}}^{\perp}, {\sigma^{\dagger}}^{\perp}, B^{\dagger}}{{\Gamma^{\dagger}}^{\perp}, \sigma^{\dagger} \multimap B^{\dagger}}$$

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

$$\mathsf{I}_i\,\frac{}{\vdash \mathsf{I} : \mathbf{1}} \qquad \mapsto \qquad \mathbf{1}\,\frac{}{\mathbf{1}}$$

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

$$\otimes_i\,\frac{\Gamma \vdash M : \sigma \quad \Delta \vdash N : \tau}{\Gamma, \Delta \vdash M \otimes N : \sigma \otimes \tau} \qquad \mapsto \qquad \otimes\,\frac{{\Gamma^{\dagger}}^{\perp}, \sigma^{\dagger} \quad {\Delta^{\dagger}}^{\perp}, \tau^{\dagger}}{{\Gamma^{\dagger}}^{\perp}, {\Delta^{\dagger}}^{\perp}, \sigma^{\dagger} \otimes \tau^{\dagger}}$$

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

$$\forall_i\,\frac{\Gamma \vdash M : A}{\Gamma \vdash M : \forall X.A} \qquad \mapsto \qquad \forall\,\frac{{\Gamma^{\dagger}}^{\perp}, A^{\dagger}}{{\Gamma^{\dagger}}^{\perp}, \forall X.A^{\dagger}}$$

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

$$\mathsf{f!p}\,\frac{\Gamma \vdash M : \sigma}{!\Gamma \vdash M : !\sigma} \qquad \mapsto \qquad \mathsf{f!p}\,\frac{{\Gamma^{\dagger}}^{\perp}, \sigma^{\dagger}}{!{\Gamma^{\dagger}}^{\perp}, !\sigma^{\dagger}}$$

$$\mathsf{?w}\,\frac{\Gamma \vdash M : \tau}{\Gamma, x : !\sigma \vdash M : \tau} \qquad \mapsto \qquad \mathsf{?w}\,\frac{{\Gamma^{\dagger}}^{\perp}, \tau^{\dagger}}{{\Gamma^{\dagger}}^{\perp}, ?{\sigma^{\dagger}}^{\perp}, \tau^{\dagger}}$$

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

FIGURE 17. Translation from PTA_2 to PLL_2.

$$\mathsf{stream}\ \frac{\vdash \mathbf{M}(0):\sigma \quad \vdash \mathbf{M}(1):\sigma \quad \dots \quad \vdash \mathbf{M}(n):\sigma \quad \dots}{\vdash \mathbf{M}:\omega\sigma} \quad\mapsto\quad \mathsf{stream}\ \frac{\sigma^\dagger \quad \sigma^\dagger \quad \dots \quad \sigma^\dagger \quad \dots}{!\sigma^\dagger}$$

$$\mathsf{discard}\ \frac{}{\vdash \mathsf{discard}:\sigma\otimes\omega\sigma \multimap \sigma} \quad\mapsto\quad \invamp\ \dfrac{?\mathsf{w}\ \dfrac{\mathsf{ax}\ \dfrac{}{\sigma^{\dagger^\perp},\sigma^\dagger}}{\sigma^{\dagger^\perp}, ?\sigma^{\dagger^\perp}, \sigma^\dagger}}{\sigma^{\dagger^\perp} \invamp ?\sigma^{\dagger^\perp}, \sigma^\dagger}$$

$$\mathsf{pop}\ \frac{}{\vdash \mathsf{pop}:\omega\sigma \multimap \sigma\otimes\omega\sigma} \quad\mapsto\quad ?\mathsf{w}\ \dfrac{\otimes\ \dfrac{\mathsf{ax}\ \dfrac{}{\sigma^{\dagger^\perp},\sigma^\dagger} \qquad \mathsf{f!p}\ \dfrac{\mathsf{ax}\ \dfrac{}{\sigma^{\dagger^\perp},\sigma^\dagger}}{?\sigma^{\dagger^\perp}, !\sigma^\dagger}}{\sigma^{\dagger^\perp}, ?\sigma^{\dagger^\perp}, \sigma^\dagger\otimes !\sigma^\dagger}}{?\sigma^{\dagger^\perp}, \sigma^\dagger\otimes !\sigma^\dagger}$$

FIGURE 18. Translation from nuPTA_2 to nuPLL_2.

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

$$\otimes\ \dfrac{\dfrac{\vdots}{\vdash \mathbf{M}(0):\sigma} \qquad \mathsf{stream}\ \dfrac{\vdash \mathbf{M}(1):\sigma \quad \vdash \mathbf{M}(2):\sigma \quad \dots \quad \vdash \mathbf{M}(n+1):\sigma \quad \dots}{\vdash tl(\mathbf{M}):\omega\sigma}}{\vdash \mathbf{M}(0)\otimes tl(\mathbf{M}):\sigma\otimes\omega\sigma}$$

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

Let $M_1 = (\lambda x.P)N$ and $M_2 = P[N/x]$. The proof is by induction on $\mathcal{D}_1$. We only show the case where the last rule of $\mathcal{D}_1$ is $\multimap_e$, i.e., we have

$$\multimap_e\ \dfrac{?\mathsf{b},?\mathsf{w}\ \dfrac{\multimap_i\ \dfrac{\overset{\mathcal{D}_1'}{\Sigma, x:\sigma \vdash P':B}}{\Sigma \vdash \lambda x.P':\sigma\multimap B}}{\Gamma \vdash \lambda x.P:\sigma\multimap B} \qquad \overset{\mathcal{D}_1''}{\Delta \vdash N:\sigma}}{\Gamma,\Delta \vdash (\lambda x.P)N:B}$$

By Lemma 79 there is $S(\mathcal{D}_1', \mathcal{D}_1'')$ such that:

$$\mathsf{cut}\ \dfrac{\left(\overset{\mathcal{D}_1''}{\Delta \vdash N:\sigma}\right)^\dagger \qquad \left(\overset{\mathcal{D}_1'}{\Sigma, x:\sigma \vdash P':B}\right)^\dagger}{?\mathsf{b},?\mathsf{w}\ \dfrac{\Sigma^{\dagger^\perp}, \Delta^{\dagger^\perp}, B^\dagger}{\Gamma^{\dagger^\perp}, \Delta^{\dagger^\perp}, B^\dagger}} \quad \to^*_{\mathsf{cut}} \quad ?\mathsf{b},?\mathsf{w}\ \dfrac{\left(\overset{S(\mathcal{D}_1',\mathcal{D}_1'')}{\Sigma,\Delta \vdash P'[N/x]:B}\right)^\dagger}{\Gamma^{\dagger^\perp}, \Delta^{\dagger^\perp}, B^\dagger} \tag{9}$$

We set $\mathcal{D}_2 = S(\mathcal{D}_1', \mathcal{D}_2'')$. We notice that $\mathcal{D}_1^\dagger$ is as follows:

$$\text{cut}\,\dfrac{\parr\,\dfrac{\begin{array}{c}(\mathcal{D}_1')^\dagger\\ \vdots\\ {\Gamma^\dagger}^\perp, {\sigma^\dagger}^\perp, B^\dagger\end{array}}{{\Gamma^\dagger}^\perp, {\sigma^\dagger}^\perp \parr B^\dagger} \qquad \otimes\,\dfrac{\begin{array}{c}(\mathcal{D}_1'')^\dagger\\ \vdots\\ {\Delta^\dagger}^\perp, \sigma^\dagger\end{array} \qquad \text{ax}\,\dfrac{}{{B^\dagger}^\perp, B^\dagger}}{{\Delta^\dagger}^\perp, \sigma^\dagger \otimes {B^\dagger}^\perp, B^\dagger}}{{\Gamma^\dagger}^\perp, {\Delta^\dagger}^\perp, B^\dagger}$$

By Equation (9) we can conclude that $\mathcal{D}_1^\dagger \to^*_{\mathsf{cut}} \mathcal{D}_2^\dagger$. □

Theorem 81. *Let $f : (\{\mathbf{0},\mathbf{1}\}^*)^n \to \{\mathbf{0},\mathbf{1}\}^*$:*

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

Proof. We only show 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 $s \in \{0,1\}^*$. Moreover, let $\mathcal{D}_f$ and $\mathcal{D}_s$ be such that

$$\mathcal{D} = \;\multimap_e\,\dfrac{\begin{array}{c}\mathcal{D}_f\\ \vdots\\ \vdash \underline{f} : \mathbf{S}[] \multimap \mathbf{S}[]\end{array} \quad \begin{array}{c}\mathcal{D}_s\\ \vdots\\ \vdash \underline{s} : \mathbf{S}[]\end{array}}{\vdash \underline{f}\,\underline{s} : \mathbf{S}[]}$$

By repeatedly applying Lemma 80 there is $\mathcal{D}_{f(s)}$ such that

$$\mathcal{D}^\dagger = \;\text{cut}\,\dfrac{\begin{array}{c}\mathcal{D}_f^\dagger\\ \vdots\\ \mathbf{S}[] \multimap \mathbf{S}[]\end{array} \qquad \otimes\,\dfrac{\begin{array}{c}\mathcal{D}_s^\dagger\\ \vdots\\ \mathbf{S}[]\end{array} \qquad \text{ax}\,\dfrac{}{\mathbf{S}[]^\perp, \mathbf{S}[]}}{\mathbf{S}[] \otimes \mathbf{S}[]^\perp, \mathbf{S}[]}}{\mathbf{S}[]} \quad \to^*_{\mathsf{cut}} \quad \begin{array}{c}\mathcal{D}_{f(s)}^\dagger\\ \vdots\\ \mathbf{S}[]\end{array}$$

in nuPLL_2, where we can safely assume that $\mathcal{D}_s^\dagger \to^*_{\mathsf{cut}} \underline{s}$ and $\mathcal{D}_{f(s)}^\dagger \to^*_{\mathsf{cut}} \underline{f(s)}$ 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.e

□

7. Conclusion and future work

This paper builds on a series of recent works aimed at developing the topic of *Cyclic Implicit Complexity*, i.e., implicit computational complexity in the setting of circular and non-wellfounded proof theory [17, 19]. We presented the non-wellfounded proof systems wrPLL_2^∞ and rPLL_2^∞ inspired by Mazza's parsimonious logic [26, 27], and prove that they characterise the complexity classes $\mathbf{FP}/\mathsf{poly}$ and $\mathbf{FP}$ respectively. We investigated infinitary cut-elimination for these non-wellfiunded proof systems, and we defined a semantics based on the relational model.

For future research, we envisage extending the contributions of this paper in many directions.

- Proof systems based on linear logic and implementing polytime computation over actual binary streams have been considered, e.g., in [42], which provide an ICC-like characterisation of Ko's class of polynomial time computable functions over real numbers [43]. As a research direction, we plan to capture Ko's class in our non-wellfounded setting. This requires to introduce the co-aborption rule to model the "pop" operation on streams. Computation over real numbers will be implemented in our systems by exploiting our continuous cut-elimination theorem (**??**).

- Another direction is to explore applications of the contributions of this paper to probabilistic complexity. In particular, we aim to study fragments of wrPLL_2^∞ modelling the class **BPP** (bounded-error probabilistic polynomial time), essentially by leveraging on well-known derandomisation methods showing the inclusion of **BPP** in $\mathbf{FP}/\mathsf{poly}$, and hence in $\mathbf{FP}(\mathbb{R})$ (see Proposition 6). A challenging aspect of this task is to obtain characterisation results that are entirely in the style of ICC, since **BPP** is defined by explicit (error) bounds, as observed in [44]. We suspect that wrPLL_2^∞ represents the right framework for investigating fully implicit characterisations of this class, where additional proof-theoretic conditions can be introduced to restrict computationally the access to oracles and, consequently, to model bounded-error probabilistic computation.
- Finally, we plan to study computational counterparts to the non-welllfounded proof systems wrPLL_2^∞ and rPLL_2^∞ based on infinitary (typed) lambda calculi, along the lines of Mazza's work on infinitary affine lambda calculus [45].

Acknowledgements. We would like to thank Anupam Das, Abhishek De, Farzad Jafar-Rahmani, Alexis Saurin, Tito (Lê Thành Dung Nguyên) and Damiano Mazza for their useful comments and suggestions.

References

[1] D. Niwiński and I. Walukiewicz, "Games for the μ-calculus," *Theoretical Computer Science*, vol. 163, no. 1-2, pp. 99–116, 1996.

[2] C. Dax, M. Hofmann, and M. Lange, "A proof system for the linear time μ-calculus," in *International Conference on Foundations of Software Technology and Theoretical Computer Science*. Springer, 2006, pp. 273–284.

[3] J. Brotherston and A. Simpson, "Sequent calculi for induction and infinite descent," *Journal of Logic and Computation*, vol. 21, no. 6, pp. 1177–1216, 2011.

[4] S. Berardi and M. Tatsuta, "Classical system of Martin-Lof's inductive definitions is not equivalent to cyclic proofs," *Log. Methods Comput. Sci.*, vol. 15, no. 3, 2019. [Online]. Available: https://doi.org/10.23638/LMCS-15(3:10)2019

[5] A. Das and D. Pous, "A cut-free cyclic proof system for Kleene algebra," in *International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*. Springer, 2017, pp. 261–277.

[6] ——, "Non-Wellfounded Proof Theory For (Kleene+Action)(Algebras+Lattices)," in *27th EACSL Annual Conference on Computer Science Logic (CSL 2018)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), D. Ghica and A. Jung, Eds., vol. 119. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, pp. 19:1–19:18. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2018/9686

[7] A. Simpson, "Cyclic arithmetic is equivalent to peano arithmetic," in *Foundations of Software Science and Computation Structures - 20th International Conference, FOSSACS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, ser. Lecture Notes in Computer Science, J. Esparza and A. S. Murawski, Eds., vol. 10203, 2017, pp. 283–300. [Online]. Available: https://doi.org/10.1007/978-3-662-54458-7_17

[8] S. Berardi and M. Tatsuta, "Equivalence of inductive definitions and cyclic proofs under arithmetic," in *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS*

2017, Reykjavik, Iceland, June 20-23, 2017. IEEE Computer Society, 2017, pp. 1–12. [Online]. Available: https://doi.org/10.1109/LICS.2017.8005114

[9] A. Das, "On the logical complexity of cyclic arithmetic," *Log. Methods Comput. Sci.*, vol. 16, no. 1, 2020. [Online]. Available: https://doi.org/10.23638/LMCS-16(1:1)2020

[10] D. Baelde, A. Doumane, and A. Saurin, "Infinitary proof theory: the multiplicative additive case," vol. 62, pp. 42:1–42:17, 2016. [Online]. Available: https://doi.org/10.4230/LIPIcs.CSL.2016.42

[11] A. De and A. Saurin, "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*, ser. Lecture Notes in Computer Science, S. Cerrito and A. Popescu, Eds., vol. 11714. Springer, 2019, pp. 297–316. [Online]. Available: https://doi.org/10.1007/978-3-030-29026-9_17

[12] A. Das, "A circular version of Gödel's T and its abstraction complexity," *CoRR*, vol. abs/2012.14421, 2020. [Online]. Available: https://arxiv.org/abs/2012.14421

[13] D. Kuperberg, L. Pinault, and D. Pous, "Cyclic proofs, system T, and the power of contraction," *Proc. ACM Program. Lang.*, vol. 5, no. POPL, pp. 1–28, 2021. [Online]. Available: https://doi.org/10.1145/3434282

[14] A. Das, "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)*, ser. LIPIcs, N. Kobayashi, Ed., vol. 195. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, pp. 29:1–29:23. [Online]. Available: https://doi.org/10.4230/LIPIcs.FSCD.2021.29

[15] G. E. Mints, "Finite investigations of transfinite derivations," *Journal of Soviet Mathematics*, vol. 10, no. 4, pp. 548–596, 1978.

[16] J. Fortier and L. Santocanale, "Cuts for circular proofs: semantics and cut-elimination," in *Computer Science Logic 2013 (CSL 2013)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.

[17] G. Curzi and A. Das, "Cyclic implicit complexity," in *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science*, ser. LICS '22. New York, NY, USA: Association for Computing Machinery, 2022. [Online]. Available: https://doi.org/10.1145/3531130.3533340

[18] S. Bellantoni and S. Cook, "A new recursion-theoretic characterization of the polytime functions (extended abstract)," in *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing*, ser. STOC '92. New York, NY, USA: Association for Computing Machinery, 1992, p. 283–293. [Online]. Available: https://doi.org/10.1145/129712.129740

[19] G. Curzi and A. Das, "Non-uniform complexity via non-wellfounded proofs," in *31st EACSL Annual Conference on Computer Science Logic, CSL 2023, February 13-16, 2023, Warsaw, Poland*, ser. LIPIcs, B. Klin and E. Pimentel, Eds., vol. 252. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023, pp. 16:1–16:18. [Online]. Available: https://doi.org/10.4230/LIPIcs.CSL.2023.16

[20] S. Arora and B. Barak, *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

[21] J.-Y. Girard, "Light linear logic," *Information and Computation*, vol. 143, no. 2, pp. 175–204, 1998. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0890540198927006

[22]

[23] Y. Lafont, "Soft linear logic and polynomial time," *Theor. Comput. Sci.*, vol. 318, no. 1-2, pp. 163–180, 2004. [Online]. Available: https://doi.org/10.1016/j.tcs.2003.10.018

[24] V. Danos and J. Joinet, "and elementary time," *Inf. Comput.*, vol. 183, no. 1, pp. 123–137, 2003. [Online]. Available: https://doi.org/10.1016/S0890-5401(03)00010-5

[25] P. Baillot, "On the expressivity of elementary linear logic: Characterizing ptime and an exponential time hierarchy," *Inf. Comput.*, vol. 241, pp. 3–31, 2015. [Online]. Available: https://doi.org/10.1016/j.ic.2014.10.005

[26] D. Mazza, "Simple parsimonious types and logarithmic space," in *24th EACSL Annual Conference on Computer Science Logic, CSL 2015, September 7-10, 2015, Berlin, Germany*, ser. LIPIcs, S. Kreutzer, Ed., vol. 41. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015, pp. 24–40. [Online]. Available: https://doi.org/10.4230/LIPIcs.CSL.2015.24

[27] D. Mazza and K. Terui, “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*, ser. Lecture Notes in Computer Science, M. M. Halldórsson, K. Iwama, N. Kobayashi, and B. Speckmann, Eds., vol. 9135. Springer, 2015, pp. 350–361. [Online]. Available: https://doi.org/10.1007/978-3-662-47666-6_28

[28] D. Baelde and D. Miller, “Least and greatest fixed points in linear logic,” in *Logic for Programming, Artificial Intelligence, and Reasoning, 14th International Conference, LPAR 2007, Yerevan, Armenia, October 15-19, 2007, Proceedings*, ser. Lecture Notes in Computer Science, N. Dershowitz and A. Voronkov, Eds., vol. 4790. Springer, 2007, pp. 92–106. [Online]. Available: https://doi.org/10.1007/978-3-540-75560-9_9

[29] T. Ehrhard and F. Jafar-Rahmani, “On the denotational semantics of linear logic with least and greatest fixed points of formulas,” *CoRR*, vol. abs/1906.05593, 2019. [Online]. Available: http://arxiv.org/abs/1906.05593

[30] H. G. Mairson and K. Terui, “On the computational complexity of cut-elimination in linear logic,” in *Italian Conference on Theoretical Computer Science*, 2003.

[31] M. Gaboardi and S. Ronchi Della Rocca, “From light logics to type assignments: A case study,” *Logic Journal of the IGPL*, vol. 17, 09 2009.

[32] A. S. Troelstra and H. Schwichtenberg, *Basic Proof Theory*, 2nd ed., ser. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2000.

[33] D. Baelde, A. Doumane, and A. Saurin, “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*, ser. LIPIcs, J. Talbot and L. Regnier, Eds., vol. 62. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016, pp. 42:1–42:17. [Online]. Available: https://doi.org/10.4230/LIPIcs.CSL.2016.42

[34] M. Pagani and L. Tortora de Falco, “Strong normalization property for second order linear logic,” *Theor. Comput. Sci.*, vol. 411, no. 2, p. 410–444, jan 2010. [Online]. Available: https://doi.org/10.1016/j.tcs.2009.07.053

[35] M. Acclavio, G. Curzi, and G. Guerrieri, “Infinitary cut-elimination via finite approximations,” 2023.

[36] Y. Lafont, “Soft linear logic and polynomial time,” *Theoretical Computer Science*, vol. 318, no. 1, pp. 163–180, 2004, implicit Computational Complexity.

[37] L. Roversi and L. Vercelli, “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*, ser. EPTCS, P. Baillot, Ed., vol. 23, 2010, pp. 63–77. [Online]. Available: https://doi.org/10.4204/EPTCS.23.5

[38] G. Curzi and A. Das, “Cyclic implicit complexity,” in *LICS ’22: 37th Annual ACM/IEEE Symposium on Logic in Computer Science, Haifa, Israel, August 2 - 5, 2022*, C. Baier and D. Fisman, Eds. ACM, 2022, pp. 19:1–19:13. [Online]. Available: https://doi.org/10.1145/3531130.3533340

[39] T. Ehrhard, F. Jafarrahmani, and A. Saurin, “On relation between totality semantic and syntactic validity,” in *5th International Workshop on Trends in Linear Logic and Applications (TLLA 2021)*, Rome (virtual), Italy, Jun. 2021. [Online]. Available: https://hal-lirmm.ccsd.cnrs.fr/lirmm-03271408

[40] T. Ehrhard, “An introduction to differential linear logic: proof-nets, models and antiderivatives,” 2016.

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

[42] E. Hainry, D. Mazza, and R. Péchoux, “Polynomial time over the reals with parsimony,” in *Functional and Logic Programming - 15th International Symposium, FLOPS 2020, Akita, Japan, September 14-16, 2020, Proceedings*, ser. Lecture Notes in Computer Science, K. Nakano and K. Sagonas, Eds., vol. 12073. Springer, 2020, pp. 50–65. [Online]. Available: https://doi.org/10.1007/978-3-030-59025-3_4

[43] K.-I. Ko, *Complexity Theory of Real Functions.* USA: Birkhauser Boston Inc., 1991.

[44] U. D. Lago and P. P. Toldin, “A higher-order characterization of probabilistic polynomial time,” *Inf. Comput.*, vol. 241, pp. 114–141, 2015. [Online]. Available: https://doi.org/10.1016/j.ic.2014.10.009

[45] D. Mazza, “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, ser. Lecture Notes in Computer Science, J. Esparza, P. Fraigniaud, T. Husfeldt, and E. Koutsoupias, Eds., vol. 8573. Springer, 2014, pp. 305–317. [Online]. Available: https://doi.org/10.1007/978-3-662-43951-7_26