Enumerating Error Bounded Polytime Algorithms Through Arithmetical Theories

Anonymous author

Anonymous affiliation

Abstract

We consider a minimal extension of the language of arithmetic, such that the bounded formulas provably total in a suitably-defined theory *à la Buss* (expressed in this new language) precisely capture polytime *random* functions. Then, we provide two new characterizations of the semantic class **BPP** obtained by internalizing the error-bound check *within* a logical system: one relies on measure-sensitive quantifiers, the other captures such quantifiers by standard first-order quantification. This leads us to introduce a family of effectively enumerable subclasses of **BPP**, each called $\mathbf{BPP}_{\mathsf{T}}$, each consisting of languages captured by those probabilistic Turing machines whose underlying error can be proved bounded in the corresponding arithmetical theory T. As a paradigmatic consequence of this approach, we establish that polynomial identity testing is in $\mathbf{BPP}_{\mathsf{PA}}$.

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

Keywords and phrases Bounded Arithmetic, Randomized Computation, Implicit Computational Complexity

1 Introduction

Since the early days of computer science, numerous and profound interactions with mathematical logic have emerged (think of the seminal works by Turing [49] and Church [8]). Among the sub-fields of computer science that have benefited the most from this dialogue, we should certainly mention the theory of programming languages (e.g. through the Curry-Howard correspondence [16, 33, 48]), the theory of databases (e.g. through Codd theorem [10]) and computational complexity (e.g. through descriptive complexity [3, 34]). In particular, this last discipline deals with complexity classes [31, 9, 2], the nature of which still remains today, more than fifty years after the introduction of **P** and **NP** [11, 31], essentially mysterious.

The possibility of describing fundamental classes within the language of mathematical logic offered a better understanding of their nature: since the seventies [22, 13], but especially from the eighties and nineties [6, 28, 3, 34, 37], the logical characterization of several crucial classes has made it possible to consider them from a new viewpoint, less dependent on concrete machine models and explicit resource bounds. Characterizing complexity classes by way of a simple enough proof-or-recursion theoretical systems also means being able to *enumerate* the problems belonging to them, and thus to devise sound and complete languages for the class, from which type systems and static analysis methodologies can be derived [32].

Among the various classes of problems with which computational complexity has been concerned, those defined on the basis of *randomized* algorithms [43] have appeared difficult to capture with the tools of logic. These include important and well-studied classes like **BPP** or **ZPP**. The former, in particular, is often considered as *the* class of feasible problems, and most complexity theorists conjecture that it actually coincides with **P**. However, by simply looking at its definition, **BPP** looks pretty different from **P**. Notably, the former, but not the latter, is an example of what is usually called a *semantic* class: the definition **BPP** relies on algorithms which are both efficient and not *too erratic*; in other words, once an input is fixed, one of the two possible outputs must clearly prevail over the other, i.e. it must occur with some probability, strictly greater than one half, and independent from the input. By their very nature, semantic classes, like **BPP**, are more challenging to be logically captured with respect to other (syntactic) classes, like **P** or **PP**: indeed, *enumerating* them through the underlying machines is harder. Currently, it is simply not known whether an effective enumeration of the aforementioned randomized classes is possible. Indeed, the sparse contributions along these lines are either themselves *semantic*, i.e. do not capture the limitations on the probability of error within the logical system [21, 19] (an interesting exception being [36]), or deal with classes, as **PP**, which are not classifiable as semantic [17, 18].

In this paper we make a step towards a logical characterization of randomized classes by considering a language in which the probability of error can be kept under control *from within* the logic. We introduce a language, called $\mathcal{RL}$, inspired by the one presented in Ferreira's [24], but in which formulas can access a source of randomness through a distinguished unary predicate `Flip`, this way naturally capturing randomized algorithms. We define a bounded theory of arithmetic, called $R\Sigma^b_1$-NIA, as the randomized analogue of Buss' S^1_2 [6] and Ferreira's Σ^b_1-NIA [24], and show that the functions which can be proved total in $R\Sigma^b_1$-NIA are precisely polytime *random* functions [47], i.e. those functions from strings to *distributions* of strings which can be computed by polytime probabilistic Turing machines (PTM, for short). Using this result, we provide two characterizations of the problems in **BPP**: one relies on *measure quantifiers* [42, 40, 1], i.e. well-studied second-order quantifiers capable of measuring *the extent* to which a formula is true; the other consists in showing that these quantifiers, when applied to bounded formulas, can be encoded via *standard* first-order quantification.

Both these approaches provide precise characterizations of **BPP** but are still semantic in nature: the entropy check is translated into conditions which are not based on provability in some formal system, but rather on the truth of some formula in the standard model of first-order arithmetic. Our arithmetization of **BPP**, however, naturally leads to the introduction of a family of new *syntactic* subclasses of **BPP**, namely $\mathbf{BPP}_{\mathsf{T}}$, made of languages for which the error-bounding condition is *provable* in a (non necessarily bounded) theory T. While we consider unlikely that for some effectively enumerable theory T, $\mathbf{BPP}_{\mathsf{T}} = \mathbf{BPP}$, we show that full first-order Peano Arithmetic (PA, for short) provides an interesting candidate theory, as $\mathbf{BPP}_{\mathsf{PA}}$ includes *polynomial identity testing* (PIT), which is one of the few problems in **BPP** currently not known to be in **P**. This fact seems very promising, suggestingan avenue towards a new form of *reverse* computational complexity in the framework of first-order arithmetic.

The main technical contributions of this paper can be summarized as follows:

- We introduce the arithmetical theory $R\Sigma^b_1$-NIA and prove that the functions which are Σ^b_1-representable in it are precisely the random functions which can be computed in polynomial time. The proof of the correspondence goes through the definition of a class of *oracle recursive* functions, called $\mathcal{POR}$, which is shown equivalent to that of functions which are Σ^b_1-representable in $R\Sigma^b_1$-NIA and, then, to the class of probabilistic polytime random functions **RFP**. The overall structure of the proof is described in Section 3, while further details can be found in Appendix A and B.
- Then, in Section 4, we use the aforementioned result to obtain a new syntactic characterization of **PP** and, more interestingly, two semantic characterizations of **BPP**, the first based on measure quantifiers and the second relying on standard, first-order quantification.
- Finally, we introduce a family of syntactic subclasses $\mathbf{BPP}_{\mathsf{T}} \subseteq \mathbf{BPP}$, parametrically on a theory T. The core idea is to consider a (sound) theory T in which error-bound checks can be syntactically considered, this way potentially restricting the class of problems to be

captured. We then prove that PIT is in $\mathbf{BPP}_{\mathsf{PA}}$, thus identifying a nontrivial effectively enumerable subclass of **BPP**. We believe this to be the most interesting and impactful of the results presented in the paper. We conclude by showing how our approach relates to existing works capturing **BPP** languages in bounded arithmetic [36]. All this can be found in Sections 5 and 6.

Related Work While a recursion-theoretic characterization of the syntactic class **PP** can be found in [17], most existing characterizations of **BPP** are based on some external, semantic condition [19, 41]. In particular, Eickmeyer and Grohe [21] provide a semantic characterization of **BPP** in a logic with fixed-point operators and a special counting quantifier, associated with a probabilistic semantics not too different from the quantitative interpretation we present in Section 3. On the other hand, a different approach is provided in the previously mentioned [36] and in [35], where Jeřábek relies on bounded arithmetic to provide characterizations of (both syntactic and semantic) randomized classes, as $\mathbf{ZPP}, \mathbf{RP}$ and **coRP** (a detailed comparison between our work and Jeřábek's one is offered in Section 6). Finally, [41] defines a higher-order language for polytime oracle recursive functions based on an adaptation of Bellantoni-Cook's safe recursion.

2 On the Enumeration of Complexity Classes

Before delving into the technical details, it is worth spending a few words on the problem of enumerating complexity classes, and on the reasons why it is more difficult for semantic classes than for syntactic ones.

Ordinary syntactic classes, as **P**, **PP**, and **PSPACE**, are quite simple to enumerate. While verifying resource bounds for *arbitrary* programs is very difficult, it is surprisingly easy to define an enumeration of resource bounded algorithms containing at least *one* algorithm for any problem in one of the aforementioned classes. To clarify what we mean, suppose we want to characterize the class **P**. On the one hand, the class of *all* algorithms working in polynomial time is recursion-theoretically very hard, actually Σ^0_2-complete [30]. On the other hand, the class of those programs consisting of a `for` loop executed a polynomial number of times, whose body itself consists of conditionals and simple enough instructions manipulating string variables, is both trivial to enumerate and big enough to characterize **P**, at least in an extensional sense: every problem in this class is decided by *at least one* program in the class and vice versa. Many characterizations of **P** (and of other syntactic classes), as those based on safe-recursion [3, 39], light and soft linear logic [27, 26, 38], and bounded arithmetic [6], can be seen as instances of the just described pattern, where the precise class of polytime programs varies, while the underlying class of problems remains unchanged.

But what about semantic classes? Although the distinction between syntactic and semantic classes appears in many popular textbooks (for instance in [2, 44]), in the literature these notions are not defined in a precise way. Generally speaking, syntactic classes are presented imposing limitations on the *amount of resources* the underlying algorithm is allowed to use. Semantic classes requires an additional condition, typically expressing that the underlying algorithm returns the correct answer *often enough*. Otherwise said, being resource bounded is not sufficient for an algorithm to solve some problem in a semantic class, since there can well be algorithms getting wrong too often. This distinction between semantic classes, as **BPP**, and syntactic ones refers to *how* a class is defined and not to the underlying set of problems. It is thus of *intensional* nature.

In semantic classes, unfortunately, the enumeration strategy sketched above does not seem to be readily applicable. How can we isolate a simple enough subclass of algorithms – which are not only resource bounded, but also not too erratic – at the same time saturating the class? We think that this paper can help us understanding the nature of this problem, without giving a definite answer. We do *not* prove **BPP** to be effectively enumerable, which we believe to be unlikely, but show that there is a subclass of it which is, on the one hand, large enough to include interesting problems in **BPP** and, on the other, "syntactic enough" to be effectively enumerable.

3 From Arithmetic to Randomized Computation, Subrecursively

In this section we introduce the two main ingredients on which our characterization of polytime randomized functions rely: a randomized bounded theory of arithmetic $R\Sigma^b_1$-NIA, and a Cobham-style function algebra for polytime oracle recursive functions, called $\mathcal{POR}$.

3.1 Recursive Functions and Arithmetical Formulas

In the 1980s, *bounded theories of arithmemtic*, i.e. subsystems of PA, where only *bounded quantifications* is admitted, were shown able to characterize several complexity classes [6, 7]. At the core of these characterizations lies the fundamental result (known since Gödel's [29]) that recursive functions can be *represented* in PA by means of Σ^0_1-formulas, i.e. formulas of the form $\exists x_1.\dots.\exists x_n.A$, where A is quantifier-free. For example, the following formula

$$A(x_1, x_2, y) := \exists x_3.x_1 \times x_2 = x_3 \land y = succ(x_3)$$

represents the function $f(x_1, x_2) = (x_1 \cdot x_2) + 1$. Indeed, in PA one can prove that $\forall x_1.\forall x_2.\exists! y.A(x_1, x_2, y)$, namely that A expresses a *functional* relation, and check that for all $n_1, n_2, m \in \mathbb{N}$, $A(\overline{n_1}, \overline{n_2}, \overline{m})$ holds (in the standard model $\mathcal{N}$) precisely when $m = f(n_1, n_2)$. Buss' intuition was then that, by considering theories *weaker* than PA, it is even possible to capture functions computable within given resource bounds [6, 7].

We aim to extend this approach also to classes of *randomized* computable functions. Our strategy focuses on a simple correspondence between first-order predicates over natural numbers and *oracles* from the Cantor space. Indeed, suppose the aforementioned recursive function f has now the ability to observe an infinite sequence of bits. For instance, f might observe the first bit of the infinite oracle string and if this bit is 0, it returns $(x_1 \cdot x_2) + 1$; otherwise, it returns 0. Our idea is that we can capture the call by f to the oracle by adding to the standard language of PA a new unary predicate `Flip`, to be interpreted as a stream of (random) bits. Our function f can then be represented by the following formula:

$$B(x_1, x_2, y) := \big(\mathtt{Flip}(\overline{0}) \land \exists x_3.x_1 \times x_2 = x_3 \land y = succ(x_3)\big) \lor \big(\neg\mathtt{Flip}(\overline{0}) \land y = \overline{0}\big)$$

As in the case above, it is possible to prove that $B(x_1, x_2, y)$ is functional, that is, that $\forall x_1.\forall x_2.\exists! y.B(x_1, x_2, y)$. However, since B now contains the unary predicate symbol `Flip`, the actual numerical function that B represents depends on the choice of an interpretation for `Flip`, i.e. on the choice of an oracle for f.

The rest of the section is devoted to the presentation of the key ingredients of this correspondence, which will be made precise in Section 3.3.

The Language $\mathcal{RL}$. In the following, we let $\mathbb{B} := \{\mathbf{0}, \mathbf{1}\}$, $\mathbb{S} := \mathbb{B}^*$ indicate the set of finite words from $\mathbb{B}$, and $\mathbb{O} := \mathbb{B}^{\mathbb{S}}$ denote the set of infinite ones. Our first goal is to introduce a language for first-order arithmetic incorporating the new predicate symbol $\mathtt{Flip}(x)$ and its interpretation in the standard model. Following [25], we consider a first-order signature for natural numbers *in binary notation*. Consistently, formulas will be interpreted over $\mathbb{S}$ rather than $\mathbb{N}$.[1]

▶ **Definition 1.** The terms and formulas of $\mathcal{RL}$ are defined by the grammars below:

$$t, s ::= x \mid \epsilon \mid \mathtt{0} \mid \mathtt{1} \mid t \frown s \mid t \times s$$

$$F, G ::= \mathtt{Flip}(t) \mid t = s \mid t \subseteq s \mid \neg F \mid F \wedge G \mid F \vee G \mid \exists x.F \mid \forall x.F.$$

The function symbol $\frown$ stands for string concatenation, while $t \times u$ indicates the concatenation of t with itself a number of times corresponding to the length of u. The binary predicate $\subseteq$ stands for the initial substring relation.[2] As standard, we let $A \to B := \neg A \vee B$.

For readability we use the following abbreviations: ts for $t \frown s$; $\mathtt{1}^t$ for $\mathtt{1} \times t$; $t \preceq s$ for $\mathtt{1}^t \subseteq \mathtt{1}^s$, expressing that the length of t is smaller than that of s; $t|_r = s$ for $(\mathtt{1}^r \subseteq \mathtt{1}^t \wedge s \subseteq t \wedge \mathtt{1}^r = \mathtt{1}^s) \vee (\mathtt{1}^t \subseteq \mathtt{1}^r \wedge s = t)$, expressing that s is the *truncation* of t at the length of r. For each string $\sigma \in \mathbb{S}^*$, we let $\overline{\sigma}$ indicate the term of $\mathcal{RL}$ representing it (so that, $\overline{\epsilon} = \epsilon$, $\overline{\sigma\mathbf{0}} = \overline{\sigma}\mathtt{0}$ and $\overline{\sigma\mathbf{1}} = \overline{\sigma}\mathtt{1}$).

As for standard bounded arithmetics [6, 23], a defining feature of our theory is the focus on so-called *bounded quantifications*. In $\mathcal{RL}$, *bounded quantifications* are of the forms $\forall x.\mathtt{1}^x \subseteq \mathtt{1}^t \to F$ and $\exists x.\mathtt{1}^x \subseteq \mathtt{1}^t \wedge F$, abbreviated as $\forall x \preceq t.F$ and $\exists x \preceq t.F$. Following [23], we adopt *subword quantifications* as those quantifications of the forms $\forall x.(\exists w \subseteq t.wx \subseteq t) \to F$ and $\exists x.\exists w \subseteq t.wx \subseteq t \wedge F$, abbreviated as $\forall x \subseteq^* t.F$ and $\exists x \subseteq^* t.F$. An $\mathcal{RL}$-formula F is said to be a *bounded* Σ*-formula* (in short, Σ^b_1) if it is of the form $\exists x_1 \preceq t_1. \dots .\exists x_n \preceq t_n.G$, where the only quantifications in G are subword ones. The distinction between bounded and subword quantifications is relevant for complexity reasons: if $\sigma \in \mathbb{S}$ is a string of length k, the witness of a subword existentially quantified formula $\exists y.y \subseteq^* \overline{\sigma} \wedge H$ is to be looked for among all possible sub-strings of σ, i.e. within a space of size $\mathcal{O}(k)$, while the witness of a bounded formula $\exists y \preceq \overline{\sigma}.H$ is to be looked for among all possible strings of length k, i.e. within a space of size $\mathcal{O}(2^k)$.

The Borel Semantics of $\mathcal{RL}$. We introduce a *quantitative* semantics for formulas of $\mathcal{RL}$, inspired by [1]. The main intuition behind this semantics is that, while the function symbols of $\mathcal{RL}$, as well as the predicate symbols "=" and "$\subseteq$", have a standard interpretation as relations over $\mathbb{S}$, the predicate symbol $\mathtt{Flip}$ may stand for *an arbitrary* predicate over $\mathbb{S}$, that is, an arbitrarily chosen $\omega \in \mathbb{O}$. For this reason, it makes sense to take as the interpretation of an $\mathcal{RL}$-formula F the set $[\![F]\!] \subseteq \mathbb{O}$ of *all* possible interpretations of $\mathtt{Flip}$ satisfying F. Importantly, such sets $[\![F]\!]$ can be proved to be *measurable*, a fact that will turn out essential in Section 4. Indeed, the canonical first-order model of $\mathcal{RL}$ over $\mathbb{S}$ can be extended to a probability space $(\mathbb{O}, \sigma(\mathcal{C}), \mu)$ defined in a standard way: here $\sigma(\mathcal{C}) \subseteq \wp(\mathbb{O})$ is the Borel σ-algebra generated by *cylinders* $\mathsf{C}^b_\sigma = \{\omega \mid \omega(\sigma) = \mathbf{b}\}$, with $\mathbf{b} \in \mathbb{B}$ (corresponding to

[1] Observe that working with strings is not crucial and all results below could be spelled out in terms of natural numbers. Indeed, theories have been introduced in both formulations, namely Ferreira's Σ^b_1-NIA and Buss' S^1_2, and proved equivalent [25].

[2] In order to avoid misunderstanding, let us briefly sum up the different notions and symbols used for subword relations. We will use $\subseteq$ to express the relation between strings, i.e. $x \subseteq y$ expresses that x is an initial or prefix substring of y. As said, we use $\subseteq$ as a relation symbol in the language $\mathcal{RL}$. We will use $\preceq$ as an auxiliary symbol in the language of $\mathcal{RL}$; in particular $t \preceq s$ is syntactic sugar for $\mathtt{1}^t \subseteq \mathtt{1}^s$. We will use $\subseteq^*$ as an auxiliary symbol in the language $\mathcal{RL}$ to denote the subword relation in general (not necessarily initial subword).

$b \in \{0,1\}$), and the measure μ is uniquely defined by the condition $\mu(\mathsf{C}^b_\sigma) = \frac{1}{2}$ (see [4]). The interpretation for terms in $\mathcal{RL}$ is as predictable, while semantics for formulas is defined below.

▶ **Definition 2** (Borel Semantics of $\mathcal{RL}$)**.** Given a term t, a formula F and an environment $\xi : \mathcal{G} \to \mathbb{S}$, where $\mathcal{G}$ is the set of term variables, the *interpretation of F under ξ* is the measurable set $[\![F]\!]_\xi \in \sigma(\mathcal{C})$ inductively defined as follows:

$$[\![t = s]\!]_\xi := \begin{cases} \mathbb{O} & \text{if } [\![t]\!]_\xi = [\![s]\!]_\xi \\ \emptyset & \text{otherwise} \end{cases} \qquad [\![\mathtt{Flip}(t)]\!]_\xi := \{\omega \mid \omega([\![t]\!]_\xi) = \mathbf{1}\} \qquad [\![\exists x.G]\!]_\xi := \bigcup_{i \in \mathbb{S}} [\![G]\!]_{\xi\{x \leftarrow i\}}$$

$$[\![\neg G]\!]_\xi := \mathbb{O} - [\![G]\!]_\xi$$

$$[\![t \subseteq s]\!]_\xi := \begin{cases} \mathbb{O} & \text{if } [\![t]\!]_\xi \subseteq [\![s]\!]_\xi \\ \emptyset & \text{otherwise} \end{cases} \qquad [\![G \vee H]\!]_\xi := [\![G]\!]_\xi \cup [\![H]\!]_\xi \qquad [\![\forall x.G]\!]_\xi := \bigcap_{i \in \mathbb{S}} [\![G]\!]_{\xi\{x \leftarrow i\}}.$$

$$[\![G \wedge H]\!]_\xi := [\![G]\!]_\xi \cap [\![H]\!]_\xi$$

This semantics is well-defined as the sets $[\![\mathtt{Flip}(t)]\!]_\xi$, $[\![t = s]\!]_\xi$ and $[\![t \subseteq s]\!]_\xi$ are measurable and measurability is preserved by all the logical operators.

Observe that an interpretation of the language $\mathcal{RL}$, in the usual first-order sense, is given by some ξ (defined as above) plus the choice of an interpretation ω for $\mathtt{Flip}(x)$. One can easily check by induction that, for any formula F and interpretation ξ, $\omega \in [\![F]\!]_\xi$ precisely when $\xi + \omega \vDash F$ in the first-order sense.

The Bounded Theory $R\Sigma^b_1$-NIA. We now introduce a bounded theory in the language $\mathcal{RL}$, called $R\Sigma^b_1$-NIA, which can be seen as a probabilistic counterpart to Ferreira's Σ^b_1-NIA [23]. The theory $R\Sigma^b_1$-NIA is defined by axioms belonging to two classes:

- *Basic axioms* (where $\mathtt{b} \in \{\mathtt{0}, \mathtt{1}\}$):

$$x\epsilon = x \qquad x \times \epsilon = \epsilon \qquad x \subseteq \epsilon \leftrightarrow x = \epsilon \qquad x\mathtt{b} = y\mathtt{b} \to x = y$$

$$x(y\mathtt{b}) = (xy)\mathtt{b} \qquad x \times y\mathtt{b} = (x \times y)x \qquad x \subseteq y\mathtt{b} \leftrightarrow x \subseteq y \vee x = y\mathtt{b} \qquad x\mathtt{0} \neq y\mathtt{1} \qquad x\mathtt{b} \neq \epsilon$$

- *Axiom schema for induction on notation*: $B(\epsilon) \wedge \forall x.\big(B(x) \to B(x\mathtt{0}) \wedge B(x\mathtt{1})\big) \to \forall x.B(x)$, where B is a Σ^b_1-formula in $\mathcal{RL}$.

The axiom schema for induction on notation adapts the usual induction schema of PA to the binary representation. The restriction to Σ^b_1-formulas is essential to characterize algorithms with *bounded* resources. Indeed, more general instances of this schema would lead to represent functions which are not polytime computable.

3.2 An Algebra of Polytime Oracle Recursive Functions

We now introduce a Cobham-style function algebra, called $\mathcal{POR}$, for polytime *oracle* recursive functions, and show that it is captured by a class of bounded formulas provably representable in the theory $R\Sigma^b_1$-NIA. This algebra is inspired by Ferreira's $\mathcal{PTCA}$ [23, 24]. Yet, a fundamental difference is that the functions we define are of the form $f : \mathbb{S}^k \times \mathbb{O} \to \mathbb{S}$, i.e. they carry an additional argument $\omega : \mathbb{S} \to \mathbb{B}$, to be interpreted as the underlying source of random bits. Furthermore, our class include the basic *query* function, which can be used to observe any bit from ω.

The *class* $\mathcal{POR}$ is the smallest class of functions from $\mathbb{S}^k \times \mathbb{O}$ to $\mathbb{S}$, containing the *empty* function $E(x,\omega) = \boldsymbol{\epsilon}$, the *projection* functions $P^n_i(x_1, \ldots, x_n, \omega) = x_i$, the *word-successor* function $S_b(x,\omega) = x\mathbf{b}$, the *conditional* function $C(\boldsymbol{\epsilon}, y, z_0, z_1, \omega) = y$ and $C(x\mathbf{b}, y, z_0, z_1, \omega) = z_b$, where $\mathbf{b} \in \mathbb{B}$ (corresponding to $b \in \{0,1\}$), the *query* function $Q(x,\omega) = \omega(x)$, and closed under the following schemata:

- *Composition*, where f is defined from $g, h_1, \ldots, h_k$ as $f(\vec{x},\omega) = g(h_1(\vec{x},\omega), \ldots, h_k(\vec{x},\omega), \omega)$.

- *Bounded recursion on notation*, where f is defined from g, h_0, h_1 as

$$f(\vec{x}, \boldsymbol{\epsilon}, \omega) = g(\vec{x}, \omega);$$
$$f(\vec{x}, y\mathbf{0}, \omega) = h_0\big(\vec{x}, y, f(\vec{x}, y, \omega), \omega\big)|_{t(\vec{x},y)};$$
$$f(\vec{x}, y\mathbf{1}, \omega) = h_1\big(\vec{x}, y, f(\vec{x}, y, \omega), \omega\big)|_{t(\vec{x},y)},$$

  with t obtained from $\boldsymbol{\epsilon}, \mathbf{0}, \mathbf{1}, \frown, \times$ by explicit definition, i.e. by applying $\frown$ and $\times$ on constants $\boldsymbol{\epsilon}$, $\mathbf{0}$, $\mathbf{1}$, and variables $\vec{x}$ and y.[3]

We now show that functions of $\mathcal{POR}$ are precisely those which are Σ^b_1-representable in $R\Sigma^b_1$-NIA. To do so, we slightly modify Buss' representability conditions by adding a constraint relating the quantitative semantics of formulas in $\mathcal{RL}$ and the additional functional parameter ω of oracle recursive functions.

▶ **Definition 3.** A function $f : \mathbb{S}^k \times \mathbb{O} \to \mathbb{S}$ is Σ^b_1*-representable in* $R\Sigma^b_1$*-NIA* if there exists a Σ^b_1-formula $G(\vec{x}, y)$ of $\mathcal{RL}$ such that:

1. $R\Sigma^b_1$-NIA $\vdash \forall\vec{x}.\exists!y.G(\vec{x}, y)$,
2. for all $\sigma_1, \dots, \sigma_k, \tau \in \mathbb{S}$ and $\omega \in \mathbb{O}$, $f(\sigma_1, \dots, \sigma_k, \omega) = \tau$ iff $\omega \in [\![G(\overline{\sigma_1}, \dots \overline{\sigma_k}, \overline{\tau})]\!]$.

Observe that Condition 1. of Definition 3 does *not* say that the unique value y is obtained as a function of inputs $\vec{x}$ *only*. Indeed, the truth-value of a formula depends both on the value of its first-order variables and on the value assigned to the random predicate `Flip`. Hence Condition 1. actually says that y is uniquely determined as a function *both* of its first-order inputs and of an oracle from $\mathbb{O}$, precisely as functions of $\mathcal{POR}$. Indeed, the following correspondence holds.

▶ **Theorem 1.** *For any function* $f : \mathbb{S}^k \times \mathbb{O} \to \mathbb{S}$*,* f *is* Σ^b_1*-representable in* $R\Sigma^b_1$*-NIA iff* $f \in \mathcal{POR}$*.*

Proof sketch. ($\Leftarrow$) The desired Σ^b_1-formula is constructed by induction on the structure of oracle recursive functions. Observe that the formula $\forall\vec{x}.\exists!y.G(\vec{x}, y)$ occurring in Condition 1. of Definition 3 is *not* Σ^b_1, since it is universally quantified while the existential quantifier is not bounded. Hence, in order to apply the inductive steps (corresponding to functions defined by composition and bounded recursion on notation), we need to adapt Parikh's theorem [45] (which holds for S^1_2) to $R\Sigma^b_1$-NIA, to state that if $R\Sigma^b_1$-NIA $\vdash \forall\vec{x}.\exists y.G(\vec{x}, y)$, where $G(\vec{x}, y)$ is a Σ^b_1-formula, then we can find a term t such that $R\Sigma^b_1$-NIA $\vdash \forall\vec{x}.\exists y \preceq t.G(\vec{x}, y)$. ($\Rightarrow$) The proof consists in adapting Cook and Urquhart's argument for system IPV^ω [12], and this goes through a *realizability interpretation* of the intuitionistic version of $R\Sigma^b_1$-NIA, called $IR\Sigma^b_1$-NIA. Further details can be found in the Appendix A. ◀

3.3 Characterizing Polytime Random Functions

Theorem 1 shows that it is possible to characterize $\mathcal{POR}$ by means of a system of bounded arithmetic. Yet, this is not enough to deal with classes, as **BPP** or **RP**, which are defined in terms of functions computed by a PTM. Observe that there is a crucial difference in the way in which probabilistic machines and oracle recursive functions access randomness, so our next goal is to fill the gap, by relating these classes of functions.

Let $\mathbb{D}(\mathbb{S})$ indicate the set of *distributions over* $\mathbb{S}$, that is, those functions $\lambda : \mathbb{S} \to [0, 1]$ such that $\sum_{\sigma\in\mathbb{S}} \lambda(\sigma) = 1$. By a *random function* we indicate a function of the form $f : \mathbb{S} \to \mathbb{D}(\mathbb{S})$.

[3] Notice that this language is identical to the one of $\mathcal{RL}$ terms, see Definition 1.

Observe that any polytime PTM $\mathcal{M}$ computes a random function $f_{\mathcal{M}}$, where, for every $\sigma_1, \dots, \sigma_k, \tau \in \mathbb{S}$, $f_{\mathcal{M}}(\sigma_1, \dots, \sigma_k)(\tau)$ coincides with the probability that $\mathcal{M}(\sigma_1 \sharp \dots \sharp \sigma_k) \Downarrow \tau$. However, a random function needs not be computed by a PTM in general. We define the following class of *polytime random functions*:

▶ **Definition 4** (Class **RFP**). The class **RFP** is made of all random functions $f : \mathbb{S}^k \to \mathbb{D}(\mathbb{S})$ such that $f = f_{\mathcal{M}}$, for some PTM $\mathcal{M}$ running in polynomial time.

Observe that functions of **RFP** are closed by *monadic composition* $\diamond$, where $(g \diamond f)(\sigma)(\tau) = \sum_{\rho \in \mathbb{S}} g(\rho)(\tau) \cdot f(\sigma)(\rho)$ (indeed, one can easily check $f_{\mathcal{M}'} \diamond f_{\mathcal{M}} = f_{\mathcal{M}' \circ \mathcal{M}}$, where $\circ$ indicates PTM composition).

Since functions of **RFP** have a different shape from those of $\mathcal{POR}$, we must adapt the notion of Σ^b_1-representability for them, relying on the fact that any closed $\mathcal{RL}$-formula F generates a *measurable* set $[\![F]\!] \subseteq \mathbb{B}^{\mathbb{N}}$.

▶ **Definition 5.** A function $f : \mathbb{S}^k \to \mathbb{D}(\mathbb{S})$ is *Σ^b_1-representable in $R\Sigma^b_1$-NIA* if there exists a Σ^b_1-formula $G(\vec{x}, y)$ of $\mathcal{RL}$ such that:

1. $R\Sigma^b_1$-NIA $\vdash \forall \vec{x}.\exists! y.G(\vec{x}, y)$,
2. for all $\sigma_1, \dots, \sigma_k, \tau \in \mathbb{S}$, $f(\sigma_1, \dots, \sigma_k, \tau) = \mu\big([\![G(\overline{\sigma_1}, \dots, \overline{\sigma}_k, \overline{\tau})]\!]\big)$.

Notice that any Σ^b_1-formula $G(\vec{x}, y)$ satisfying Condition 1. from Definition 5 actually defines a random function $\langle G \rangle : \mathbb{S} \to \mathbb{D}(\mathbb{S})$ given by $\langle G \rangle(\vec{\sigma})(\tau) = \mu([\![G(\overline{\vec{\sigma}}, \overline{\tau})]\!])$, where $\langle G \rangle$ is Σ^b_1-represented by G. Moreover, if G represents some $f \in$ **RFP**, then $f = \langle G \rangle$. In analogy with Theorem 1, we can now prove the following result:

▶ **Theorem 2.** *For any function $f : \mathbb{S}^k \to \mathbb{D}(\mathbb{S})$, f is Σ^b_1-representable in $R\Sigma^b_1$-NIA iff $f \in$ **RFP**.*

Thanks to Theorem 1, the proof of the result above simply consists in showing that $\mathcal{POR}$ and **RFP** can be related as stated below.

▶ **Lemma 3.** *For all functions $f : \mathbb{S}^k \times \mathbb{O} \to \mathbb{S}$ in $\mathcal{POR}$ there exists $g : \mathbb{S}^k \to \mathbb{D}(\mathbb{S})$ in **RFP** such that for all $\sigma_1, \dots, \sigma_k, \tau \in \mathbb{S}$, $\mu(\{\omega \mid f(\vec{\sigma}, \omega) = \tau\}) = g(\sigma_1, \dots, \sigma_k, \tau)$, and conversely.*

Proof sketch. The proof is rather convoluted. The first step consists in replacing the class **RFP** by an intermediate class **SFP** corresponding to functions computed by polytime *stream Turing machines* (STM, for short). These are defined as deterministic TM with one extra read-only tape: at the beginning of the computation the extra tape is sampled from $\mathbb{B}^{\mathbb{N}}$, and at each computation step the machine reads one new bit from this tape. Then we show that for any function $f : \mathbb{S}^k \to \mathbb{D}(\mathbb{S})$ computed by some polytime PTM there is a function $g : \mathbb{S}^k \times \mathbb{B}^{\mathbb{N}} \to \mathbb{S}$ computed by a polytime STM such that for all $\sigma_1, \dots, \sigma_k, \tau \in \mathbb{S}$, and $\eta \in \mathbb{B}^{\mathbb{N}}$, $f(\sigma_1, \dots, \sigma_k, \tau) = \mu(\{\eta \mid g(\sigma_1, \dots, \sigma_k, \eta) = \tau\})$, and conversely. To conclude, we prove the correspondence between the classes $\mathcal{POR}$ and **SFP**:

(SFP $\Rightarrow \mathcal{POR}$) The encoding relies on the remark that, given an input $x \in \mathbb{S}$ and an extra-tape $\eta \in \mathbb{B}^{\mathbb{N}}$, an STM $\mathcal{S}$ running in polynomial time can only access a *finite* portion of η, bounded by some polynomial $p(|x|)$. This way the behavior of $\mathcal{S}$ is encoded by a $\mathcal{POR}$-function $h(x, y)$, where the second string y corresponds to $\eta_{p(|x|)}$, and we can define $f^\sharp(x, \omega) = h(x, e(x, \omega))$, where $e : \mathbb{S} \times \mathbb{O} \to \mathbb{S}$ is a function of $\mathcal{POR}$ which mimics the prefix extractor $\eta \mapsto \eta_{p(|x|)}$, in the sense that its outputs have the same distributions of all possible η's prefixes (yet over $\mathbb{O}$ rather than $\mathbb{B}^{\mathbb{N}}$).

($\mathcal{POR} \Rightarrow$ SFP) Here we must consider that these two models not only invoke oracles of different shape, but also that functions of $\mathcal{POR}$ can manipulate such oracles in a much more liberal way than STMs. Notably, the STM accesses oracle bits in a *linear* way: each bit is used exactly once and cannot be re-invoked. Moreover, at each step of computation the STM queries a new oracle bit, while functions of $\mathcal{POR}$ can access the oracle, so to say, *on demand*. The argument rests then on a chain of simulations, making use of a class of imperative languages inspired by Winskell's IMP [50], each one taking care of one specific oracle access policy: first non-linear and on-demand (as for $\mathcal{POR}$), then linear but still on-demand, and finally linear and not on-demand (as for STMs).

For further details, see the Appendix B. ◀

4 Towards BPP

We now turn our attention to randomized complexity classes. This requires us to consider how random functions (and thus PTMs) correspond to languages, namely subsets of $\mathbb{S}$. First observe that the language computed by a random function can naturally be defined via a majority rule:

▶ **Definition 6.** Let $f : \mathbb{S} \to \mathbb{D}(\mathbb{S})$ be a random function. The language $\mathrm{Lang}(f) \subseteq \mathbb{S}$ is defined by $\sigma \in \mathrm{Lang}(f)$ iff $f(\sigma)(\epsilon) > \frac{1}{2}$.

As a warm-up, let us take a look at how **PP** can be characterized. Clearly, a language L is in **PP** precisely when $L = \mathrm{Lang}(f)$, for some polytime random function $f \in \mathbf{RFP}$. Using Theorem 2, this readily leads then to a characterization of **PP** within $R\Sigma^b_1$-NIA.

▶ **Proposition 4** (Syntactic Characterization of **PP**)**.** *For any language $L \subseteq \mathbb{S}$, $L \in \mathbf{PP}$ iff there is a Σ^b_1-formula $G(x, y)$ such that:*
1. *$R\Sigma^b_1$-NIA $\vdash \forall x \exists! y.G(x, y)$,*
2. *$L = \mathrm{Lang}(\langle G \rangle)$.*

The characterization above is syntactic as it provides an enumeration of **PP** (by enumerating the pairs made of a formula G and a proof in $R\Sigma^b_1$-NIA satisfying Condition 1). While a majority rule is enough to capture the problems in **PP**, a semantic class like **BPP** requires a stronger uniformity condition.

▶ **Definition 7** (**BPP**)**.** Given a language $L \subseteq \mathbb{S}$, $L \in \mathbf{BPP}$ iff there is a polynomial time PTM $\mathcal{M}$ such that for any $\sigma \in L$, $\Pr[\mathcal{M}(\sigma) = \chi_L(\sigma)] \geq \frac{2}{3}$ (where, $\chi_L : \mathbb{S} \to \{0, 1\}$ is the characteristic function of L).

The class **BPP** can be captured by "non-erratic" probabilistic algorithms, i.e. such that, for a fixed input, one possible output is definitely more likely than the others.

▶ **Definition 8.** A random function $f : \mathbb{S} \to \mathbb{D}(\mathbb{S})$ is said *non-erratic* if for all $\sigma \in \mathbb{S}$, $f(\sigma)(\tau) \geq \frac{2}{3}$ holds for some value $\tau \in \mathbb{S}$.

▶ **Lemma 5.** *For any language $L \subseteq \mathbb{S}$, $L \in \mathbf{BPP}$ iff $L = \mathrm{Lang}(f)$, for some non-erratic function of* **RFP** *f.*

Proof. For any non-erratic **RFP**-function f, let $\mathcal{M}$ be the PTM computing $k \diamond f$, where $k(\epsilon) = 1$ and $k(\sigma \neq \epsilon) = 0$; then $\mathcal{M}$ computes $\chi_{\mathrm{Lang}(f)}$ with error $\leq \frac{1}{3}$. Conversely, if $L \in \mathbf{BPP}$, let $\mathcal{M}$ be a PTM accepting L with error $\leq \frac{1}{3}$; then $L = \mathrm{Lang}(h \diamond f_{\mathcal{M}})$, where $h(1) = \epsilon$ and $h(\sigma \neq 1) = 0$. ◀

Lemma 5 suggests that, in order to characterize **BPP** in the spirit of Proposition 4, a new condition has to be added, corresponding to the fact that G represents a non-erratic random function. A natural question is whether a theory like $R\Sigma_1^b$-NIA or some of its extensions can be used to capture not only the resource bounds but also this new probabilistic condition.

We will show that this is actually the case. In the rest of this section we discuss two approaches to measure error bounds for probabilistic algorithms, leading to two different characterizations of **BPP**: first via measure quantifiers [1], then by purely arithmetical means. While both such methods ultimately consists in showing that the *truth* of a formula in the standard model of $R\Sigma_1^b$-NIA, they suggest a proof-theoretic approach, that we explore in the Section 5.

4.1 BPP via Measure Quantifiers

As we have seen, any $\mathcal{RL}$-formula F is associated with a measurable set $[\![F]\!] \subseteq \mathbb{O}$. So, a natural idea, already explored in [1], consists in enriching $\mathcal{RL}$ with *measure-quantifiers* [42, 40], that is, second-order quantifiers of the form $\mathbf{C}^q F$, where $q \in [0,1] \cap \mathbb{Q}$, intuitively expressing that the measure of $[\![F]\!]$ is greater than (or equal to) q. Then, let $\mathcal{RL}^{\mathsf{MQ}}$ be the extension of $\mathcal{RL}$ with measure-quantified formulas $\mathbf{C}^{t/s}F$, where t, s are terms. The Borel semantics of $\mathcal{RL}$ naturally extends to $\mathcal{RL}^{\mathsf{MQ}}$ letting:

$$[\![\mathbf{C}^{t/s}F]\!]_\xi := \begin{cases} \mathbb{O} & \text{if } |[\![s]\!]_\xi| > 0 \text{ and } \mu([\![F]\!]_\xi) \geq \frac{|[\![t]\!]_\xi|}{|[\![s]\!]_\xi|} \\ \emptyset & \text{otherwise.} \end{cases}$$

To improve readability, for all $n, m \in \mathbb{N}$, we abbreviate $\mathbf{C}^{1^n/1^m}F$ as $\mathbf{C}^{n/m}F$.

Measure quantifiers can now be used to express that the formula representing a random function is non-erratic, as shown below.

▶ **Theorem 6** (First Semantic Characterization of **BPP**). *For any language $L \subseteq \mathbb{S}$, $L \in$ **BPP** iff there is a Σ_1^b-formula $G(x, y)$ such that:*

1. $R\Sigma_1^b$*-NIA* $\vdash \forall x \exists! y.G(x, y)$,
2. $\vDash \forall x.\exists y.\mathbf{C}^{2/3}G(x, y)$,
3. $L = \mathrm{Lang}(\langle G \rangle)$.

Proof. Suppose $L \in$ **BPP** and let $g : \mathbb{S} \to \mathbb{D}(\mathbb{S})$ be a function of **RFP** computing L with uniform error bound (which, thanks to Lemma 5, we can suppose to be non-erratic). By Theorem 2, there is a Σ_1^b-formula $G(x, y)$ such that $g = \langle G \rangle$. In particular, for all $\sigma \in \mathbb{S}$, $\mu([\![G(\overline{\sigma}, \overline{\tau})]\!]) = g(\sigma)(\tau) \geq \frac{2}{3}$ holds for some $\tau \in \mathbb{S}$, which shows that Condition 2. holds. Conversely, if Conditions 1.-3. hold, then $\langle G \rangle$ computes L with the desired error bound, so $L \in$ **BPP**. ◀

4.2 Arithmetizing Measure Quantifiers

Theorem 6 relies on the tight correspondence between arithmetic and probabilistic computation; yet, Condition 2. involves formulas which are not in the language of first-order arithmetics. Lemma 7 below shows that measure quantifications over bounded formulas of $\mathcal{RL}$ can be expressed in the arithmetical language.

▶ **Lemma 7.** *For any Σ_1^b-formula $F(\vec{x})$ of $\mathcal{RL}$, there is a Σ_3^b-formula* $\mathsf{TwoThirds}[F](\vec{x})$ *such that for any $\vec{\sigma} \in \mathbb{S}$, $\vDash \mathsf{TwoThirds}[F](\overline{\vec{\sigma}})$ holds whenever $\mu([\![F(\overline{\vec{\sigma}})]\!]) \geq \frac{2}{3}$.*

Proof Sketch. In the construction of $\mathsf{TwoThirds}[F](\vec{x}, y)$ we exploit the formula $\exp(x, y)$ that defines the exponential function 2^x, given by $2^\epsilon = \mathbf{1}$ and $2^{\sigma b} = 2^\sigma 2^\sigma$. First, observe that for any bounded $\mathcal{RL}$-formula $F(\vec{x})$ and $\omega \in \mathbb{O}$, to check whether $\omega \in [\![F(\overline{\vec{\sigma}})]\!]$ only a *finite* portions of bits of ω has to be observed. More precisely, we construct a $\mathcal{RL}$-term $t_F(\vec{x})$ (containing 2^x), such that for any $\vec{\sigma} \in \mathbb{S}$ and $\omega, \omega' \in \mathbb{O}$, if $\omega|_{|t_F(\vec{\sigma})|} = \omega'|_{|t_F(\vec{\sigma})|}$, then $\omega \in [\![F(\sigma)]\!]$ when $\omega' \in [\![F(\sigma)]\!]$. Using this fact, measuring $[\![F(\sigma)]\!]$ is reduced to *counting* the elements of a finite set of strings of length $|t_F(\vec{\sigma})|$. This set can be captured by transforming $F(\vec{x})$ into a formula $\mathsf{NoFlip}[F](\vec{x}, y)$, intuitively expressing that y is a string of length $|t_F(\vec{x})|$ encoding the initial segment of ω satisfying $F(\vec{x})$, and this way eliminating all the occurrences of $\mathtt{Flip}(x)$. Using so-called *threshold quantifiers* $\exists^{\geq t_F} x.F$, the original formula is then converted into $\exists^{\geq u_{2/3}} y.\mathsf{NoFlip}[F](\vec{x}, y)$, where u is a large-enough term corresponding to the $\frac{2}{3}$ probability requirement. Finally, we show that threshold quantification $\exists^{\geq w} y.F(\vec{x}, y)$ over a Σ^b_1-formula can be encoded in $\mathcal{RL}$ via the Σ^b_3-formula $\exists y \preceq u_F.\mathsf{Threshold}[F](\vec{x}, y, w)$. So, we conclude defining $\mathsf{TwoThirds}[F](\vec{x}) := \exists y \preceq u_{\mathsf{NoFlip}[F]}.\mathsf{Threshold}[\mathsf{NoFlip}[F](\vec{x}, y)](\vec{x}, y, u)$. ◀

Theorem 6 and Lemma 7 together yield a purely arithmetical characterization of **BPP**.

▶ **Theorem 8** (Second Semantic Characterization of **BPP**). *For any language $L \subseteq \mathbb{S}$, $L \in$ **BPP** when there is a Σ^b_1-formula $G(x, y)$ such that:*
1. $R\Sigma^b_1$*-NIA* $\vdash \forall x.\exists! y.G(x, y)$,
2. $\models \forall x.\exists y.\mathsf{TwoThirds}[G](x, y)$,
3. $L = \mathrm{Lang}(\langle G \rangle)$.

5 On Syntactic Subclasses of BPP

The characterization provided by Theorem 8 is still semantic in nature, as it provides no way to effectively enumerate **BPP**. Indeed, the crucial Condition 2 is not checked within a formal system, but over the standard model of $R\Sigma^b_1$-NIA. Yet, since the condition is now expressed in purely arithmetical terms, it makes sense to consider *syntactic* variants of Condition 2, where the check on $\mathbb{S}$ is replaced by provability in some sufficiently expressive theory.

We will work in extensions of Σ^b_1-RNIA + Exp, where Exp is the formula expressing the totality of 2^x (which is used in the de-randomization of Lemma 7). This naturally leads to the following definition:

▶ **Definition 9** (Class $\mathbf{BPP}_\mathsf{T}$). Let $\mathsf{T} \supseteq \Sigma^b_1$-RNIA + Exp be a theory in the language $\mathcal{RL}$. The *class* **BPP** *relative to* T, namely $\mathbf{BPP}_\mathsf{T}$, contains all languages $L \subseteq \mathbb{S}$ such that for some Σ^b_1-formula $G(x, y)$ the following hold:
1. $R\Sigma^b_1$-NIA $\vdash \forall x.\exists! y.G(x, y)$,
2. $\mathsf{T} \vdash \forall x.\exists y.\mathsf{TwoThirds}[G](x, y)$,
3. $L = \mathrm{Lang}(\langle G \rangle)$.

Whenever T is sound (i.e. $\mathsf{T} \vdash F$ implies that F is true in the standard model), it is clear that $\mathbf{BPP}_\mathsf{T} \subseteq \mathbf{BPP}$. However, a crucial difference between the *syntactic* class $\mathbf{BPP}_\mathsf{T}$ and **BPP** is that, when T is recursively enumerable, $\mathbf{BPP}_\mathsf{T}$ can be *enumerated* (by enumerating the proofs of Condition 1. and 2. in T), while, as we discussed above, it does not seem possible to define an enumeration of randomized algorithms containing a witness for any problem in **BPP**, at least not *directly*. For this reason, it is unlikely that one can find a sound r.e. theory T such that $\mathbf{BPP}_\mathsf{T} = \mathbf{BPP}$.

In the rest of this section, we investigate how far we can go in this direction, namely whether there is any sound theory T such that the corresponding class $\mathbf{BPP}_\mathsf{T}$ looks more similar to **BPP** than to, say, **P**. A fundamental observation is that, while the restriction to bounded theories is crucial to capture polytime algorithms by a totality condition (i.e. Condition 1.), there is no reason to restrict ourselves to such theories to prove probabilistic algorithms to be non-erratic (i.e. Condition 2.). For this reason in the following we take full PA as our candidate theory T, and show that polynomial identity testing (PIT) is provably **BPP** in this theory.

5.1 Polynomial Zero Testing is in $\mathbf{BPP}_{\mathsf{PA}}$

In this subsection we pave the way to the proof that PIT is in $\mathbf{BPP}_{\mathsf{PA}}$. Here we take PA as formulated in the sub-language of $\mathcal{RL}$, where the symbol $\mathbf{0}$ is omitted, each natural number n is represented by the term $\mathbf{1}^n$, and the successor is interpreted as $\cdot \frown \mathbf{1}$.

The PIT problem asks to decide the identity of the polynomial computed by two arithmetical circuits. These are basically DAGs whose nodes can be labeled so as to denote an input, an output, the constants $0, 1$ or an arithmetic operation. These structures can easily be encoded, e.g. using lists, as terms of PA.

▶ **Definition 10** (cf. [2]). The problem PIT asks to decide whether two arithmetical circuits p, q encoded as lists of nodes describe the same polynomial, i.e. $\mathbb{Z} \models p = q$.

Usually, PIT is reduced to another problem: the so-called Polynomial Zero Testing (PZT) problem, which asks to decide whether a polynomial computing a circuit over $\mathbb{Z}$ is zero, i.e. to check whether $\mathbb{Z} \models p = 0$. Indeed, $\mathbb{Z} \models p = q$ if and only if $\mathbb{Z} \models p - q = 0$. Our proof of the fact that the language PZT is in $\mathbf{BPP}_{\mathsf{PA}}$ is structured as follows:

- We identify a Σ^b_1-formula of $\mathcal{RL}$ G characterizing the polytime algorithm `PZT` proposed in [2] for solving PZT.
- We define a `Flip`-less Σ^b_3-formula $H(x, y)$ which represents the naïve deterministic algorithm for PZT.
- We turn G into a `Flip`-less equivalent formula of $\mathcal{RL}^{\exp}$ G^v using the procedure described in Section 4, and show that $\mathsf{PA} \vdash \forall x.\forall y.\mathsf{TwoThirds}[G(x, y) \leftrightarrow H(x, y)]$.

Observe that from the last step, using the totality of H (i.e. $\mathsf{PA} \vdash \forall x.\exists! y.H(x, y)$), one can deduce $\mathsf{PA} \vdash \forall x.\exists y.\mathsf{TwoThirds}[G](x, y)$, as required in Definition 9.

Each of the aforementioned steps will be described in one of the forthcoming subsections, although the details are discussed in the Appendix C.

The Randomized Algorithm Intuitively, our algorithm for PZT takes an input x, which encodes a circuit p of size m on the variables $v_1, \dots, v_n$, it draws $r_1, \dots, r_n$ uniformly at random from $\{0, \dots, 2^{m+3} - 1\}$ and k from $\{1, \dots, 2^{2m}\}$, then it computes the value of $p(r_1, \dots, r_n) \bmod k$, so to ensure that during the evaluation no overflow can take place. This is done linearly many times in $|x|$ (we call this value s), as to ensure that, if the polynomial is not identically zero, the probability to evaluate p on values witnessing this property at least once grows over $\frac{2}{3}$.[4] Finally, if all the evaluations returned 0 as output the input is accepted; otherwise, it is rejected.

As we discuss in Appendix C.2, the procedure described above is correct only when the size of the input circuit x is greater than some constant ϱ. If this is not the case, our algorithm queries a table T storing all the pairs $(x_i, \chi_{\mathrm{PZT}}(x_i))$ for $|x_i| < \varrho$, to obtain $\chi_{\mathrm{PZT}}(x_i)$. The table T can be pre-computed, having just a constant number of entries. This algorithm, which we call `PZT`, is inspired by [2] and can be expressed as follows:

1. If the input x is not the output of a circuit, reject it. Otherwise, let n be its arity, d its degree and m its size. Set i to 1.

[4] A suitable value of s is shown in the Appendix C.1.

2. Check whether m is smaller than some constant value $\varrho \in \mathbb{N}$.
   - If so, walk the table T looking for a pair (x_j, y_j) where $x_j = x$; set $o_i = 0$ if $y_j = 1$, set $o_i = 1$, otherwise.
   - Otherwise, proceed as follows. Choose $r_1, \ldots, r_n$ uniformly and independently in $\{0, \ldots, 2^{m+3} - 1\} \subseteq \mathbb{N}$. Let k be a random value in $\{1, \ldots, 2^{2m}\} \subseteq \mathbb{N}$. Finally, evaluate the result of x, seen as a circuit, on $r_1, \ldots, r_n \mod k$, with result o_i.
3. If $i < s$, then increase i by 1 and go back to 2.
4. If for all i, $1 \leq i \leq s$, $o_i = 0$ output ϵ; otherwise, output $\mathbf{0}$.

Because of the evaluation of the input circuit modulo some $k \in \mathbb{Z}$, this algorithm works in time polynomial with respect to $|x|$. Therefore, as a consequence of Theorem 1 and Lemma 3, there is a Σ_1^b-formula $G(x, y)$ of $\mathcal{RL}$ which characterizes it. A lower-level description of the formula G is presented in Appendix C.1.

The Underlying Language We show that there is a PA-definable predicate H such that $H(x, \epsilon)$ holds if and only if x is the encoding of a circuit in PZT; otherwise, $H(x, \mathbf{0})$ holds. This predicate realizes the function h described by the following algorithm:

1. Take in input x, and check whether it is a polynomial circuit with one output; if it is not, reject it. Otherwise:
2. Compute the polynomial term p represented by x, and reduce it to a normal form $\overline{p}$.
3. Check whether all the coefficients of the terms are null. If this is true, output ϵ, otherwise output $\mathbf{0}$ and terminate.

For reasonable encodings of polynomial circuits and expressions, h is primitive recursive and therefore there is a PA-definable predicate H which characterizes it. Moreover $h = \chi_{\mathrm{PZT}}$, indeed:

▶ Remark 9. For every polynomial p with coefficients in $\mathbb{Z}$, it holds that $\mathbb{Z} \models \forall \vec{x}.p(\vec{x}) = 0$ if and only if all the monomials in the normal form of p have zero as coefficient.

Proving the Error Bound We now show that, within PA, it is possible to prove that the formula G is not erratic and that it decides $\mathrm{Lang}(\langle G \rangle)$, as required by Definition 9, for $\mathsf{T} = \mathsf{PA}$. This can be reduced to showing that:

$$\mathsf{PA} \vdash \forall x.\forall y.\mathsf{TwoThirds}[G(x, y) \leftrightarrow H(x, y)], \tag{$*$}$$

To do so, we assume to have an encoding of finite sets — and set-theoretic predicates, such as belonging, size, etc. — as terms and predicates of PA. This allows us to reduce bounded threshold-existential quantifiers to statements on the size of finite sets. For instance, Claim $(*)$ is equivalent to the following one:

$$\mathsf{PA} \vdash \forall x.\forall y.3 \cdot |\{z \in \mathbb{B}^{t(x))} \mid \mathsf{NoFlip}[G](x, y, z) \leftrightarrow H(x, y))\}| \geq 2^{t(x)+1},$$

where $\mathsf{NoFlip}(G)$ is a formula equivalent to G, which uses z as a source of randomness, and t is a polynomially-sized term depending on x only. Further details on these objects can be found in Appendix C.2. We proceed by showing two intermediate claims:

$$\mathsf{PA} \vdash \forall z.|z| = t(x) \to \mathsf{NoFlip}[G](x, \mathbf{0}, z) \to H(x, \mathbf{0}), \tag{1}$$

which assesses that whenever the randomized algorithm rejects an input, then so does the deterministic one, and

$$\mathsf{PA} \vdash \forall x.\forall y.3 \cdot |\{z \in \mathbb{B}^{t(x))} \mid \mathsf{NoFlip}[G](x, \epsilon, z) \to H(x, \epsilon))\}| \geq 2^{t(x)+1} \tag{2}$$

which asserts that if the randomized algorithm accepts the circuit, the probability that the deterministic one accepts the language is higher than $\frac{2}{3}$. Claim (1) is a consequence of the compatibility of the mod k function with addition and multiplication, which can be proved in PA.

While Claim (1) shows that the `PZT` algorithm always accepts members of PZT, Claim (2) determines a bound to the entropy of the algorithm. The proof of the latter claim is more articulated and relies on the proofs in PA of the Schwartz-Zippel Lemma, providing a lower bound to the probability of evaluating the polynomial on values witnessing that it is not identically zero, and the Prime Number Theorem which bounds the probability to choose a *bad* value for k, i.e. one of those values causing `PZT` to return the wrong value. While the formalization in PA of the Prime Number Theorem is known [15], a precise formulation and a proof of the latter results are reported in the Appendix C.2. Conditions (1) and (2) entail Conditions (2), (3) of Definition 9 for $\mathsf{T} = \mathsf{PA}$, and thus PZT $\in \mathbf{BPP}_{\mathsf{PA}}$.

Closure of $\mathbf{BPP}_{\mathsf{PA}}$ under Polytime Reduction Only assessing that a problem belongs to $\mathbf{BPP}_{\mathsf{PA}}$ does not tell us anything about other languages of this class; for this reason, we are interested in showing that $\mathbf{BPP}_{\mathsf{PA}}$ is closed under polytime reduction. This allows us to start form PZT $\in \mathbf{BPP}_{\mathsf{PA}}$ to conclude that all problems which can be reduced to PZT in polynomial time belong to this class, not only that PIT $\in \mathbf{BPP}_{\mathsf{PA}}$. This is assessed by the following proposition:

▶ **Proposition 1.** For every language $L \in \mathbf{BPP}_{\mathsf{PA}}$ and every language $M \subseteq \mathbb{S}$, if there is a polytime reduction from M to L, then $M \in \mathbf{BPP}_{\mathsf{PA}}$.

A detailed proof of this result is given in Appendix C.4. This has as Corollary the main claim of this section:

▶ **Corollary 1.** PIT is in $\mathbf{BPP}_{\mathsf{PA}}$.

6 On Jeřábek's Characterization of BPP

As mentioned in the Section 1, a semantic characterization of **BPP** due to bounded arithmetic was already provided by Jeřábek in [36]. This approach relies on checking, against the standard model, the truth of a formula which, however, does not express an entropy condition, but can be seen as a second totality condition (beyond the formula expressing the totality of the algorithm). Hence, also in this case one can investigate which problems can be proved to be in **BPP** within some given theory.

In this section, we relate the two approaches by showing that the problems in $\mathbf{BPP}_{\mathsf{T}}$ are provably definable **BPP** problems, in the sense of [36], *within some suitable extension* of the bounded theory PV_1[12]. Indeed, Jeřábek focuses on the theory PV_1, extended with an axiom schema dWPHP called the *dual weak pigeonhole principle* (cf. [36, pp. 962ff.]), which turns out useful in counting arguments.

A PTM is represented in this setting by two provably total functions (A, r), where the machine accepts on input x with probability less than p/q when $\Pr_{w<r(x)}(A(x,w)) \leq p/q$. This formula, corresponding to asking that the set $\{w < r(x) \mid \vDash A(x,w)\}$ has cardinality smaller than $p/q \cdot r(x)$, can be formalized in the language of PV_1. The representation of **BPP** problems hinges on the definition, for any probabilistic algorithm (A, r), of $L^+_{A,r}(x) := \Pr_{w<r(x)}(\neg A(x,w)) \leq 1/3$ and $L^-_{A,r}(x) := \Pr_{w<r(x)}(A(x,w)) \leq 1/3$. Checking if the algorithm (A, r) solves some problem in **BPP** reduces then to checking the "totality" formula $\vDash \forall x. L^+_{A,r}(x) \vee L^-_{A,r}(x)$.

Now, first observe that, modulo an encoding of strings via numbers, everything which is provable in $R\Sigma^b_1$-NIA *without* the predicate `Flip` can be proved in the theory $S^1_2(PV)$ [12], which extends both PV_1 and Buss' S^1_2. Moreover, by arguing as in the proof of Lemma 5, in our characterization of $\mathbf{BPP}$ we can w.l.o.g. suppose that the formula G satisfies $\mathrm{EpsZero}[G] := \forall x.\forall y.G(x,y) \rightarrow y = \epsilon \vee y = \mathbf{0}$. Under this assumption, the derandomization procedure described in the proof of Lemma 7 turns G into a pair (A, r), where $A = \mathsf{noFlip}[G]$ and $r(x) = t_G(x)$, and the languages $L^+_{A,r}(x)$ and $L^-_{A,r}(x)$ correspond to the formulas $L^+_G(x) := \mathsf{TwoThirds}[G(-,\epsilon)](x)$, and $L^-_G(x) := \mathsf{TwoThirds}[G(-,\mathbf{0})](x)$. Notice that, contrarily to [36], $t_G(x)$ needs not be a polynomial term, since it may contain 2^x. Yet this makes little difference since we work in extensions of Σ^b_1-RNIA + Exp, i.e. in theories which capture *more* than polytime computation.

Observing that, from $\mathsf{T} \vdash \forall x.\exists y.\mathsf{TwoThirds}[G](x,y)$ and $\mathrm{EpsZero}[G]$, one can deduce $\mathsf{T} \vdash \forall x.L^+_G(x) \vee L^-_G(x)$, we arrive at the following:

▶ **Proposition 10.** *If $L \in \mathbf{BPP}_\mathsf{T}$, with $L = \mathrm{Lang}(\langle G\rangle)$, then $\forall x.L^+_G(x) \vee L^-_G(x)$ is provable in some recursively enumerable extension of PV_1+dWPHP.*

7 Conclusion

The logical characterization of randomized complexity classes, in particular those having a semantic nature, is a great challenge. This paper contributes to the understanding of this problem by showing not only how resource bounded randomized computation can be captured within the language of arithmetic, but also that the latter offers convenient tools to control error bounds, the essential ingredient in the definition of classes like $\mathbf{BPP}$ and $\mathbf{ZPP}$.

We believe that the main contribution of this work is a first example of a sort of *reverse* computational complexity for probabilistic algorithms. As we discussed in Section 5, while the restriction to bounded theories is crucial in order to capture polytime algorithms via a totality condition, it is not necessary to prove error bounds for probabilistic (even polynomial time) algorithms. Actually, since it is unlikely that $\mathbf{BPP} = \mathbf{BPP}_\mathsf{T}$ for some sound r.e. theory T, it is worth exploring how much can be proved within expressive arithmetical theories. For this reason we focused here on the problems which can be proved to be in $\mathbf{BPP}$ in full PA, and we showed that PIT is among them. Actually, we conjecture that our whole argument can be formalized in the fragment $I\Delta_0 + \mathrm{Exp}$ of PA, with induction restricted to Δ_0-formulas plus the totality of the exponential function.

Indeed, an exciting direction is the study of the expressiveness of the new syntactic classes $\mathbf{BPP}_\mathsf{T}$, that is, an investigation on the kinds of error bounds which can be proved in the arithmetical theories lying *between* standard bounded theories like S^1_2 or PV and PA, but also in theories which are more expressive than PA (like e.g. second-order theories). Given the tight connections between bounded arithmetics and proof complexity, another natural direction is the study of applications of our work to probabilistic approaches in this field, for example recent investigations on *random resolution refutations* [36, 5, 46], i.e. resolution systems where proofs may make errors but are correct most of the time. These problems, intriguing as they are, are anyway left to future work.

References

1. M. Antonelli, U. Dal Lago, and P. Pistone. On Measure Quantifiers in First-Order Arithmetic. In L. De Mol, A. Weiermann, F. Manea, and D. Fernández-Duque, editors, *Connecting with Computabiliy*, pages 12–24. Springer, 2021.

2 S. Arora and B. Barak. *Computational complexity: A Modern Approach*. Cambridge University Press, 2009.

3 S. Bellantoni and S. Cook. A New Recursion-Theoretic Characterization of the Polytime Functions. *Computational Complexity*, 2:97–110, 1992.

4 P. Billingsley. *Probability and Measure*. Wiley, 1995.

5 S. Buss, A.L. Kolodziejczyk, and N. Thapen. Fragments of Approximate Counting. *Journal of Symbolic Logic*, 79(2):496–525, 2014.

6 S.R. Buss. *Bounded Arithmetic*. PhD thesis, Princeton University, 1986.

7 S.R. Buss. First-Order Proof Theory of Arithmetic. In S.R: Buss, editor, *Handbook of Proof Theory*. Elsavier, 1998.

8 A. Church. An Unsolvable Problem of Elementary Number Theory. *American J. of Mathematics*, 58(2):345–363, 1992.

9 A. Cobham. The Intrinsic Computational Difficulty of Functions. In Y. Bar-Hillel, editor, *Logic, Methodology and Philosophy of Science: Proc. of the 1964 International Congress (Studies in Logic and the Foundations of Mathematics)*, pages 24–30. North-Holland Publishing, 1965.

10 E.F. Codd. Relational Completeness of Data Base Sublanguages. In *Data Base Systems. Proc. of 6th Courant Computer Science Symposium, May 24-25, 1971, New York, N.Y.*, pages 65–98, 1972.

11 S. Cook. The Complexity of Theorem Proving Procedures. In *Proc. Third Annual ACM Symposium on Theory of Computing*, pages 151–158, 1971.

12 S. Cook and A. Urquhart. Functional Interpretations of Feasibly Constructive Arithmetic. *Annals of Pure and Applied Logic*, 63(2):103–200, 1993.

13 S.A. Cook and R.A. Reckhow. Efficiency of Propositional Proof Systems. *Journal of Symbolic Logic*, 44(1):36–50, 1979.

14 M. Coquand, T. andd Hofmann. A new method for establishing conservativity of classical systems over their intuitionistic version. In *Proc. Mathematical Structures in Computer Science*, pages 323–333, 1999.

15 C. Cornaros and C. Dimitracopoulos. The Prime Number Theorem and Fragments of PA. *Archive for Mathematical Logic*, 33:265–281, 08 1994. `doi:10.1007/BF01270626`.

16 H. B. Curry. Functionality in Combinatory Logic*. *Proc. of the National Academy of Sciences*, 20(11):584–590, 1934.

17 U. Dal Lago, R. Kahle, and I. Oitavem. A Recursion-Theoretic Characterization of the probabilistic Class PP. In F. Bonchi and S.J. Puglisi, editors, *46th International Symposium on Mathematical Foundations of Computer Science (MFCS 2021)*, volume 202 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 1–12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

18 U. Dal Lago, R. Kahle, and I. Oitavem. Implicit Recursion-Theoretic Characterizations of Counting Classes. *Archive for Mathematical Logic*, May 2022.

19 U. Dal Lago and P. Parisen Toldin. A Higher-Order Characterization of Probabilistic Polynomial Time. *Information and Computation*, 241:114–141, 2015.

20 D. Davoli. Bounded Arithmetic and Randomized Computation. Master's thesis, University of Bologna, 2022. URL: `http://amslaurea.unibo.it/26234/`.

21 K. Eickmeyer and M. Grohe. Randomisation and Derandomisation in Descriptive Complexity Theory. In A. Dawar and H. Veith, editors, *Computer Science Logic*. Springer Berlin Heidelberg, 2010.

22 R. Fagin. Generalized First-Order Spectra and Polynomial-Time Recognizable Sets. In *Complexity of Computing: SIAM-AMS Proc.*, pages 43–73, 1974.

23 F. Ferreira. Polyonmial-Time Computable Arithmetic and Conservative Extesions. Ph.D. Dissertation, December 1988.

24 F. Ferreira. Polynomial-Time Computable Arithmetic. In W. Sieg, editor, *Logic and Computation*, volume 106 of *Contemporary Mathematics*, pages 137–156. AMS, 1990.

25 G. Ferreira and I. Oitavem. An Interpretation of S_2^1 in Σ_1^b-NIA. *Portugaliae Mathematica*, 63:427–450, 2006.

26 J.-Y. Girard. Light Linear Logic. *Information and Computation*, 2(143):175–204, 1998.

27 J.-Y. Girard and Y. Lafont. *Advances in Linear Logic.* Cambridge University Press, 1995.

28 J.-Y. Girard, A. Scedrov, and P. Scott. Bounded Linear Logic: A Modular Approach to Polynomial-Time Computability. *Theoretical Computer Science*, 97(1):1–66, 1992.

29 K. Gödel. Über Formal Unentscheidbare Sätze der Principia Mathematica and Verwandter Systeme. *Monatshefte für Mathematik und Physik*, 38:173–198, 1931.

30 P. Hajek. Arithmetical Hierarchy and Complexity of Computation. *Theoretical Computer Science*, 8(2):227–237, 1979.

31 J. Hartmanis and R.E. Stearns. On the Computational Complexity of Algorithms. *Transactions of the AMS*, 117:285–306, 1965.

32 M. Hofmann. Programming Languages Capturing Complexity Classes. *SIGACT News*, 31(1):31–42, mar 2000.

33 H. A. Howard. The Formulae-as-Types Notion of Construction. In *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism.* Academic Press, 1980.

34 N. Immerman. *Descriptive Complexity.* Springer, 1999.

35 E. Jeřábek. Dual Weak Pigeonhole Principle, Boolean Complexity, and Derandomization. *Annals of Pure and Applied Logic*, 129(1):1–37, 2004.

36 E. Jeràbek. Approximate Counting in Bounded Arithmetic. *Journal of Symbolic Logic*, 72(3):959–993, 2007.

37 J. Krajicek and P. Pudlak. Propositional Proof Systems, the Consistency of First-Order Theories and the Complexity of Computations. *Journal of Symboic Logic*, 54(3):1063–1079, 1989.

38 Y. Lafont. Soft Linear Logic adn Polynomial Time. *Theoretical Computer Science*, 1/2(318):163–180, 2004.

39 D. Leivant. Ramified Recurrence and Computational Complexity I: Word Recurrence and Polytime. In P. Clote and J. Remmel, editors, *Feasible Mathematics II*, pages 320–343. Springer, 1995.

40 H. Michalewski and M. Mio. Measure Quantifiers in Monadic Second Order Logic. In *Proc. of Logical Foundations of Computer Science*, pages 267–282, Cham, 2016. Springer.

41 J. Mitchell, M. Mitchell, and A. Scedrov. A Linguistic Characterization of Bounded Oracle Computation and Probabilistic Polynomial Time. In *Proc. of 39th Annual Symposium on Foundations of Computer science*, pages 725–733. IEEE Computer Society, 1998.

42 C. Morgenstern. The Measure Quantifier. *Journal of Symbolic Logic*, 44(1):103–108, 1979.

43 R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, Cambridge; NY, 1995.

44 C.H. Papadimitriou. *Computational Complexity.* Pearson Education, 1993.

45 R. Parikh. Existence and Feasibility in Arithmetic. *Journal of Symbolic Logic*, 36:494–508, 1971.

46 P. Pudlak and N. Thapen. Random Resolution Refutations. *Computational Complexity*, 28:185–239, 2019.

47 E.S. Santos. Probabilistic Turing Machines and Computability. *AMS*, 22(3):704–710, 1969.

48 M.H. Sorensen and P. Urzyczyn. *Lectures on the Curry-Howard Isomorphism.* Elsevier, 2006.

49 A. Turing. On Computable Numbers, with an Application to the Entscheidungsproblem. *Proc. London Mathematical Society*, pages 2–42, 230–265, 1936-37.

50 G. Winskel. *The Formal Semantics of Programming Languages: an Introduction.* MIT press, 1993.

A Proofs from Section 3.2

Theorem 1.($\Leftarrow$). As anticipated, in order to apply inductive steps (namely, composition and bounded recursion on notation) we need to adapt Parikh's theorem [45] to $R\Sigma^b_1$-NIA.[5]

▶ **Proposition 2** ("Parikh" [45]). Let $F(\vec{x}, y)$ be a bounded $\mathcal{RL}$-formula such that $R\Sigma^b_1\text{-NIA} \vdash \forall\vec{x}.\exists y.F(\vec{x}, y)$. Then, there is a term t such that, $R\Sigma^b_1\text{-NIA} \vdash \forall\vec{x}.\exists y \preceq t(\vec{x}).F(\vec{x}, y)$.

Proof for Theorem 1($\Leftarrow$). The proof is by induction on the structure of functions in $\mathcal{POR}$.

Base Case. Each basic function is Σ^b_1-representable in $R\Sigma^b_1$-NIA.

- The empty function $f = E$ is Σ^b_1-represented in $R\Sigma^b_1$-NIA by the formula:

$$F_E(x, y) : x = x \wedge y = \epsilon.$$

  1. Existence is proved considering $y = \epsilon$. For the reflexivity of identity both $R\Sigma^b_1\text{-NIA} \vdash x = x$ and $R\Sigma^b_1\text{-NIA} \vdash \epsilon = \epsilon$ hold. So, by rules for conjunctions, we obtain $R\Sigma^b_1\text{-NIA} \vdash x = x \wedge \epsilon = \epsilon$, and conclude $R\Sigma^b_1\text{-NIA} \vdash \forall x.\exists y.(x = x \wedge y = \epsilon)$. Uniqueness is proved assuming $R\Sigma^b_1\text{-NIA} \vdash x = x \wedge z = \epsilon$. By rules for conjunction, in particular $R\Sigma^b_1\text{-NIA} \vdash z = \epsilon$, and since $R\Sigma^b_1\text{-NIA} \vdash y = \epsilon$, by the transitivity of identity, we conclude $R\Sigma^b_1\text{-NIA} \vdash y = z$.
  2. Assume $E(\sigma, \omega^*) = \tau$. If $\tau = \boldsymbol{\epsilon}$, then $[\![\overline{\sigma} = \overline{\sigma} \wedge \overline{\tau} = \epsilon]\!] = [\![\overline{\sigma} = \overline{\sigma}]\!] \cap [\![\overline{\tau} = \epsilon]\!] = \mathbb{O} \cap \mathbb{O} = \mathbb{O}$. So, for any $\omega^*, \omega^* \in [\![\overline{\sigma} = \overline{\sigma} \wedge \overline{\tau} = \epsilon]\!]$, as clearly $\omega^* \in \mathbb{O}$. If $\tau \neq \boldsymbol{\epsilon}$, then $[\![\overline{\sigma} = \overline{\sigma} \wedge \overline{\tau} = \epsilon]\!] = [\![\overline{\sigma} = \overline{\sigma}]\!] \cap [\![\overline{\tau} = \epsilon]\!] = \mathbb{O} \cap \emptyset = \emptyset$. So, for any $\omega^*, \omega^* \notin [\![\overline{\sigma} = \overline{\sigma} \vee \overline{\tau} = \epsilon]\!]$, as clearly $\omega^* \notin \emptyset$.

- Functions $f = P^n_i$, $f = S_b$ and $f = C$ are Σ^b_1-represented in $R\Sigma^b_1$-NIA by respectively the formulas:

$$F_{P^n_i}(x_1, \dots, x_n, y) : \bigwedge_{j \in J} (x_j = x_j) \wedge y = x_i,$$

$$F_{S_b}(x, y) : y = x\mathtt{b},$$

$$F_C(x, v, z_0, z_1, y) : (x = \epsilon \wedge y = v) \vee \exists x' \preceq x.(x = x'\mathtt{0} \wedge y = z_0)$$
$$\vee\, \exists x' \preceq x.(x = x'\mathtt{1} \wedge y = z_1).$$

  where $1 \leq i \leq n$, $J = \{1, \dots, n\} \setminus \{i\}$, and $\mathtt{b} \in \{\mathtt{0}, \mathtt{1}\}$ corresponding to (resp.) $b \in \{0, 1\}$. Proofs are omitted as straightforward.

- $f = Q$ is Σ^b_1-represented in $R\Sigma^b_1$-NIA by the formula:

$$F_Q(x, y) : (\mathtt{Flip}(x) \wedge y = \mathtt{1}) \vee (\neg\mathtt{Flip}(x) \wedge y = \mathtt{0}).$$

  Observe that, in this case, the proof crucially relies on the fact that oracle functions invoke *exactly one* oracle:

  1. Existence is proved by cases. If $R\Sigma^b_1\text{-NIA} \vdash \mathtt{Flip}(x)$, let $y = \mathtt{1}$. By the reflexivity of identity, $R\Sigma^b_1\text{-NIA} \vdash (\mathtt{Flip}(x) \wedge \mathtt{1} = \mathtt{1}) \vee (\neg\mathtt{Flip}(x) \wedge \mathtt{0} = \mathtt{1})$ and so,

$$R\Sigma^b_1\text{-NIA} \vdash \exists y.\big((\mathtt{Flip}(x) \wedge y = \mathtt{1}) \vee (\neg\mathtt{Flip}(x) \wedge y = \mathtt{0})\big).$$

[5] The theorem is usually presented in the context of Buss' bounded theories, as stating that given a bounded formula F in $\mathcal{L}_{\mathbb{N}}$ such that $S^1_2 \vdash \forall\vec{x}.\exists y.F$, then there is a term $t(\vec{x})$ such that also $S^1_2 \vdash \forall\vec{x}.\exists y \leq t(\vec{x}).F(\vec{x}, y)$ [6, 7]. Furthermore, due to [25], Buss' syntactic proof can be adapted to Σ^b_1-NIA in a natural way. The same result holds for $R\Sigma^b_1$-NIA, as not containing specific rules concerning $\mathtt{Flip}(\cdot)$.

If $R\Sigma^b_1$-NIA $\vdash \neg\texttt{Flip}(x)$, let $y = \texttt{0}$. By the reflexivity of identity $R\Sigma^b_1$-NIA $\vdash \texttt{0} = \texttt{0}$ holds. Thus, by the rules for conjunction, $R\Sigma^b_1$-NIA $\vdash \neg\texttt{Flip}(x) \wedge \texttt{0} = \texttt{0}$ and for disjunction, we conclude $R\Sigma^b_1$-NIA $\vdash (\texttt{Flip}(x) \wedge \texttt{0} = \texttt{1}) \vee (\neg\texttt{Flip}(x) \wedge \texttt{0} = \texttt{0})$ and so,

$$R\Sigma^b_1\text{-NIA} \vdash \exists y.\big((\texttt{Flip}(x) \wedge y = \texttt{1}) \vee (\neg\texttt{Flip}(x) \wedge y = \texttt{0})\big).$$

Uniqueness is established relying on the transitivity of identity.

2. Finally, it is shown that for every $\sigma, \tau \in \mathbb{S}$ and $\omega^* \in \mathbb{O}$, $Q(\sigma, \omega^*) = \tau$ when $\omega^* \in [\![F_Q(\overline{\sigma}, \overline{\tau})]\!]$. Assume $Q(\sigma, \omega^*) = \mathbf{1}$, which is $\omega^*(\sigma) = \mathbf{1}$,

$$\begin{aligned}
[\![F_Q(\overline{\sigma}, \overline{\tau})]\!] &= [\![\texttt{Flip}(\overline{\sigma}) \wedge \overline{\tau} = \texttt{1}]\!] \cup [\![\neg\texttt{Flip}(\overline{\sigma}) \wedge \overline{\tau} = \texttt{0}]\!] \\
&= ([\![\texttt{Flip}(\overline{\sigma})]\!] \cap [\![\texttt{1} = \texttt{1}]\!]) \cup ([\![\neg\texttt{Flip}(\overline{\sigma})]\!] \cap [\![\texttt{1} = \texttt{0}]\!]) \\
&= ([\![\texttt{Flip}(\overline{\sigma})]\!] \cap \mathbb{O}) \cup ([\![\neg\texttt{Flip}(\overline{\sigma})]\!] \cap \emptyset) \\
&= [\![\texttt{Flip}(\overline{\sigma})]\!] \\
&= \{\omega \mid \omega(\sigma) = \texttt{1}\}.
\end{aligned}$$

Clearly, $\omega^* \in [\![(\texttt{Flip}(\overline{\sigma}) \wedge \overline{\tau} = \texttt{1}) \vee (\neg\texttt{Flip}(\overline{\sigma}) \wedge \overline{\tau} = \texttt{0})]\!]$. The case $Q(\sigma, \omega^*) = \mathbf{0}$ and the opposite direction are proved in a similar way.

Inductive Case. If f is defined by composition or bounded recursion from Σ^b_1-representable functions, then f is Σ^b_1-representable in $R\Sigma^b_1$-NIA:

- *Composition.* Assume that f is defined by composition from functions $g, h_1, \dots, h_k$ so that $f(\vec{x}, \omega) = g(h_1(\vec{x}, \omega), \dots, h_k(\vec{x}, \omega), \omega)$ and that $g, h_1, \dots, h_k$ are represented in $R\Sigma^b_1$-NIA by the Σ^b_1-formulas $F_g, F_{h_1}, \dots, F_{h_k}$, respectively. By Proposition 2, there exist suitable terms $t_g, t_{h_1}, \dots, t_{h_k}$ such that (the existential part of) Condition 1. can be strengthened to $R\Sigma^b_1$-NIA $\vdash \forall\vec{x}.\exists y \preceq t_i.F_i(\vec{x}, y)$ for each $i \in \{g, h_1, \dots, h_k\}$. We conclude that $f(\vec{x}, \omega)$ is Σ^b_1-represented in $R\Sigma^b_1$-NIA by the following formula:

$$\begin{aligned}
F_f(x, y) : \exists z_1 \preceq t_{h_1}(\vec{x}). \dots \exists z_k \preceq t_{h_k}(\vec{x}).(F_{h_1}(\vec{x}, z_1) \wedge \dots F_{h_k}(\vec{x}, z_k) \\
\wedge F_g(z_1, \dots, z_k, y)).
\end{aligned}$$

  Indeed, by IH, $F_g, F_{h_1}, \dots, F_{h_k}$ are Σ^b_1-formulas. Then, also F_f is in Σ^b_1. Conditions 1.-2. are proved to hold by slightly modifying standard proofs.
- *Bounded Recursion.* Assume that f is defined by bounded recursion from g, h_0, h_1, and t, so that:

$$\begin{aligned}
f(\vec{x}, \boldsymbol{\epsilon}, \omega) &= g(\vec{x}, \omega) \\
f(\vec{x}, y\texttt{b}, \omega) &= h_i(\vec{x}, y, f(\vec{x}, y, \omega), \omega)|_{t(\vec{x}, y)},
\end{aligned}$$

  where $i \in \{0, 1\}$ and $\mathbf{b} = \mathbf{0}$ when $i = 0$ and $\mathbf{b} = \mathbf{1}$ when $i = 1$. Let g, h_0, h_1 be represented in $R\Sigma^b_1$-NIA by, respectively, the Σ^b_1-formulas F_g, F_{h_0}, and F_{h_1}. Moreover, by Proposition 2, there exist suitable terms t_g, t_{h_0}, and t_{h_1} such that the existential part of condition 1. can be strengthened to its "bounded version". Then, it can be proved that $f(\vec{x}, y)$ is Σ^b_1-represented in $R\Sigma^b_1$-NIA by the formula below:

$$\begin{aligned}
F_f(x, y) : \exists v \preceq t_g(\vec{x}) t_f(\vec{x})(y \times t(\vec{x}, y) t(\vec{x}, y)\texttt{11}).(F_{lh}(v, \texttt{1} \times y\texttt{1}) \\
\wedge \exists z \preceq t_g(\vec{x}).(F_{eval}(v, \epsilon, z) \wedge F_g(\vec{x}, z)) \\
\wedge \forall u \subset y.\exists z.(\tilde{z} \preceq t(\vec{x}, y))(F_{eval}(v, \texttt{1} \times u, z) \wedge F_{eval}(v, \texttt{1} \times u \times, \tilde{z}) \\
\wedge (u\texttt{0} \subseteq y \to \exists z_0 \preceq t_{h_0}(\vec{x}, u, z).(F_{h_0}(\vec{x}, u, z, z_0) \wedge z_0|_{t(\vec{x}, u)} = \tilde{z})) \\
\wedge (u\texttt{1} \subseteq y \to \exists z_1 \preceq t_{h_1}(\vec{x}, u, z).(F_{h_1}(\vec{x}, u, z, z_1) \wedge z_1|_{t(\vec{x}, u)} = \tilde{z})))),
\end{aligned}$$

where F_{lh} and F_{eval} are Σ^b_1-formulas defined as in [23]. Intuitively, $F_{lh}(x,y)$ states that the number of $\mathtt{1}$s in the encoding of x is yy, while $F_{eval}(x,y,z)$ is a "decoding" formula (strongly resembling Gödel's β-formula), expressing that the "bit" encoded in x as its y-th bit is z. Moreover $x \subset y$ is an abbreviation for $x \subseteq y \wedge x \neq y$. Then, this formula F_f satisfies all the requirements to Σ^b_1-represent in $R\Sigma^b_1$-NIA the function f, obtained by bounded recursion form g, h_0, and h_1. In particular, Condition 1. concerning existence and uniqueness, have already been proved to hold by Ferreira [23]. Furthermore, F_f expresses that, given the desired encoding sequence v: (i.) the $\boldsymbol{\epsilon}$-th bit of v is (the encoding of) z' such that $F_g(\vec{x}, z')$ holds, where (for IH) F_g is the Σ^b_1-formula representing the function g, and (ii.) given that for each $u \subset y$, z denotes the "bit", encoded in v at position $\mathtt{1} \times u\mathtt{1}$, then if $u\mathtt{b} \subseteq y$ (that is, if we are considering the initial substring of y the last bit of which correspond to $\mathtt{b}$), then there is a z_b such that $F_{h_b}(\vec{x}, y, z, z_b)$, where F_{h_b} Σ^b_1-represents the function f_{h_b} and the truncation of z_b at $t(\vec{x}, u)$ is precisely $\tilde{z}$, with $b = 0$ when $\mathtt{b} = \mathtt{0}$ and $b = 1$ when $\mathtt{b} = \mathtt{1}$.

◀

Theorem 1.($\Rightarrow$). The proof is obtained by adapting that by Cook and Urquhart for IPV^ω [12], and is structured as follows:

1. We define $\mathcal{POR}^\lambda$ a basic equational theory for a simply typed λ-calculus endowed with primitives corresponding to functions of $\mathcal{POR}$.
2. We introduce a first-order *intuitionistic* theory $I\mathcal{POR}^\lambda$, which extends $\mathcal{POR}^\lambda$ with the usual predicate calculus as well as an **NP**-induction schema. It is shown that $I\mathcal{POR}^\lambda$ is strong enough to prove all theorems of $IR\Sigma^b_1$-NIA.
3. We develop a realizability interpretation of $I\mathcal{POR}^\lambda$ (inside itself), showing that for any derivation of $\forall x.\exists y.F(x,y)$ (where F is a Σ^b_0-formula) one can extract a λ-term $\mathtt{t}$ of $\mathcal{POR}^\lambda$, such that $\forall x.F(x, \mathtt{t}x)$ is provable in $I\mathcal{POR}^\lambda$. From this we deduce that every function which is Σ^b_1-representable in $IR\Sigma^b_1$-NIA is in $\mathcal{POR}$.
4. We extend this result to classical $R\Sigma^b_1$-NIA showing that any Σ^b_1-formula provable in $I\mathcal{POR}^\lambda$ + Excluded Middle (EM, for short) is already provable in $I\mathcal{POR}^\lambda$.

The System $\mathcal{POR}^\lambda$.

We define an equational theory for a simply typed λ-calculus augmented with primitives for functions of $\mathcal{POR}$. Actually, these do not exactly correspond to the ones of $\mathcal{POR}$, although the resulting function algebra is proved equivalent.

▶ **Definition 11.** *Types of $\mathcal{POR}^\lambda$* are defined by the grammar below:

$$\sigma := s \;\big|\; \sigma \Rightarrow \sigma.$$

▶ **Definition 12.** *Terms of $\mathcal{POR}^\lambda$* are standard, simply typed λ-terms plus the constants:

$$\begin{aligned}
\mathsf{0}, \mathsf{1}, \epsilon &: s \\
\circ, \mathsf{Trunc} &: s \Rightarrow s \Rightarrow s \\
\mathsf{Tail}, \mathsf{Flipcoin} &: s \Rightarrow s \\
\mathsf{Cond} &: s \Rightarrow s \Rightarrow s \Rightarrow s \Rightarrow s \\
\mathsf{Red} &: s \Rightarrow (s \Rightarrow s \Rightarrow s) \Rightarrow (s \Rightarrow s \Rightarrow s) \Rightarrow (s \Rightarrow s) \Rightarrow s \Rightarrow s.
\end{aligned}$$

Intuitively, $\mathsf{Tail}(x)$ computes the string obtained by deleting the first digit of x; $\mathsf{Trunc}(x,y)$ computes the string obtained by truncating x at the length of y; $\mathsf{Cond}(x,y,z,w)$ computes the function that yields y when $x = \boldsymbol{\epsilon}$, z when $x = x'\mathbf{0}$, and w when $x = x'\mathbf{1}$; $\mathsf{Flipcoin}(x)$ indicates a random $\mathbf{0}/\mathbf{1}$ generator; Rec is the operator for bounded recursion on notation.

▶ **Notation 1.** We abbreviate $x \circ y$ as xy and being T any constant $\mathsf{Tail}, \mathsf{Trunc}, \mathsf{Cond}, \mathsf{Flipcoin}, \mathsf{Rec}$ of arity n, we indicate $\mathsf{T}\mathsf{u}_1, \dots, \mathsf{u}_n$ as $\mathsf{T}(\mathsf{u}_1, \dots, \mathsf{u}_n)$.

We also introduce the following abbreviations for composed functions:

- $\mathsf{B}(x) := \mathsf{Cond}(x, \epsilon, 0, 1)$ denotes the function computing the last digit of x.
- $\mathsf{BNeg}(x) := \mathsf{Cond}(x, \epsilon, 0, 1)$ denotes the function computing the Boolean negation of $\mathsf{B}(x)$.
- $\mathsf{BOr}(x, y) := \mathsf{Cond}(\mathsf{B}(x), \mathsf{B}(y), \mathsf{B}(y), 1)$ denotes the function that coerces x and y to Booleans and then performs the OR operation.
- $\mathsf{BAnd}(x, y) := \mathsf{Cond}(\mathsf{B}(x), \epsilon, 0, \mathsf{B}(y))$ denotes the function that coerces x and y to Booleans and then performs the AND operation.
- $\mathsf{Eps}(x) := \mathsf{Cond}(x, 1, 0, 0)$ denotes the characteristic function of "$x = \mathsf{0}$".
- $\mathsf{Bool}(x) := \mathsf{BAnd}(\mathsf{Eps}(\mathsf{Tail}(x)), \mathsf{BNeg}(\mathsf{Eps}(x)))$ denotes the characteristic function of "$x = \mathsf{0} \vee x = \mathsf{1}$".
- $\mathsf{Zero}(x) := \mathsf{Cond}(\mathsf{Bool}(x), 0, \mathsf{Cond}(x, 0, 0, 1), 0)$ denotes the characteristic function of predicate "$x = \mathsf{0}$".
- $\mathsf{Conc}(x, y)$ denotes the concatenation function defined as:

$$\mathsf{Conc}(x, \epsilon) := x \qquad \mathsf{Conc}(x, y\mathsf{b}) := \mathsf{Conc}(x, y)\mathsf{b},$$

  with $\mathsf{b} \in \{0, 1\}$.
- $\mathsf{Eq}(x, y)$ denotes the characteristic function of "$x = y$" and defined by double recursion by the equations below:

$$\mathsf{Eq}(\epsilon, \epsilon) := 1 \qquad \mathsf{Eq}(\epsilon, y\mathsf{b}) := 0$$
$$\mathsf{Eq}(x\mathsf{b}, \epsilon) = \mathsf{Eq}(x0, y1) = \mathsf{Eq}(x1, y0) := 0 \qquad \mathsf{Eq}(x\mathsf{b}, y\mathsf{b}) := \mathsf{Eq}(x, y),$$

  with $\mathsf{b} \in \{0, 1\}$.
- $\mathsf{Times}(x, y)$ denotes the function for self-concatenation, $x, y \mapsto x \times y$ and is defined by the equations below:

$$\mathsf{Times}(x, \epsilon) := \epsilon \qquad \mathsf{Times}(x, y\mathsf{b}) := \mathsf{Conc}(\mathsf{Times}(x, y), x),$$

  with $\mathsf{b} \in \{0, 1\}$.
- $\mathsf{Sub}(x, y)$ denotes the initial substring function, $x, y \mapsto S(x, y)$, and is defined by bounded recursion as follows:

$$\mathsf{Sub}(x, \epsilon) := \mathsf{Eps}(x) \qquad \mathsf{Sub}(x, y\mathsf{b}) := \mathsf{BOr}(\mathsf{Sub}(x, y), \mathsf{Eq}(x, y\mathsf{b})),$$

  with $\mathsf{b} \in \{0, 1\}$.

▶ **Definition 13.** *Formulas of $\mathcal{POR}^\lambda$* are equations $\mathsf{t} = \mathsf{u}$, where t and u are terms of type s.

▶ **Definition 14** (Theory $\mathcal{POR}^\lambda$)**.** Axioms of $\mathcal{POR}^\lambda$ are the following ones:

- Defining equations for the constants of $\mathcal{POR}^\lambda$:

$$\epsilon x = x\epsilon = x \qquad x(y\mathsf{b}) = (xy)\mathsf{b}$$
$$\mathsf{Tail}(\epsilon) = \epsilon \qquad \mathsf{Tail}(x\mathsf{b}) = x$$
$$\mathsf{Trunc}(x, \epsilon) = \mathsf{Trunc}(\epsilon, x) = \epsilon \qquad \mathsf{Trunc}(x\mathsf{b}, y\mathsf{b}) = \mathsf{Trunc}(x, y)\mathsf{b}$$
$$\mathsf{Cond}(\epsilon, y, z, w) = y \qquad \mathsf{Cond}(x0, y, z, w) = z \qquad \mathsf{Cond}(x1, y, z, w) = w$$
$$\mathsf{Bool}(\mathsf{Flipcoin}(x)) = 1$$
$$\mathsf{Rec}(x, h_0, h_1, k, \epsilon) = x \qquad \mathsf{Rec}(x, h_0, h_1, k, y\mathsf{b}) = \mathsf{Trunc}(h_b y(\mathsf{Rec}(x, h_0, h_1, k, y)), ky),$$

where $\mathsf{b} \in \{0,1\}$ and $b \in \{0,1\}$ (correspondingly).

- The (β)- and (ν)-axioms:

$$\mathsf{C}[(\lambda x.\mathsf{t})\mathsf{u}] = \mathsf{C}[\mathsf{t}\{\mathsf{u}/x\}] \qquad (\beta)$$

$$\mathsf{C}[\lambda x.\mathsf{t}x] = \mathsf{C}[\mathsf{t}]. \qquad (\nu)$$

where $\mathsf{C}[\cdot]$ indicates a context with a unique occurrence of the hole [ ], so that $\mathsf{C}[\mathsf{t}]$ denotes the variable capturing replacement of [ ] by t in $\mathsf{C}[\ ]$.

The inference rules of $\mathcal{POR}^\lambda$ are the following ones:

$$\mathsf{t} = \mathsf{u} \vdash \mathsf{t} = \mathsf{u} \qquad (\mathsf{R1})$$

$$\mathsf{t} = \mathsf{u}, \mathsf{u} = \mathsf{v} \vdash \mathsf{t} = \mathsf{v} \qquad (\mathsf{R2})$$

$$\mathsf{t} = \mathsf{u} \vdash \mathsf{v}\{\mathsf{t}/x\} = \mathsf{v}\{\mathsf{u}/x\} \qquad (\mathsf{R3})$$

$$\mathsf{t} = \mathsf{u} \vdash \mathsf{t}\{\mathsf{v}/x\} = \mathsf{u}\{\mathsf{v}/x\}. \qquad (\mathsf{R4})$$

As predictable, $\vdash_{\mathcal{POR}^\lambda} \mathsf{t} = \mathsf{u}$ expresses that the equation $\mathsf{t} = \mathsf{u}$ is deducible using instances of the axioms above plus inference rules $(\mathsf{R1}) - (\mathsf{R4})$. Similarly, given any set T of equations, $T \vdash_{\mathcal{POR}^\lambda} \mathsf{t} = \mathsf{u}$ expresses that the equation $\mathsf{t} = \mathsf{u}$ is deducible using instances of the quoted axioms and rules together with equations from T.

For any string $\sigma \in \mathbb{S}$, let $\overline{\overline{\sigma}} : s$ denote the term of $\mathcal{POR}^\lambda$ corresponding to it, that is:

$$\overline{\overline{\boldsymbol{\epsilon}}} = \epsilon \qquad \overline{\overline{\sigma\mathbf{0}}} = \overline{\overline{\sigma}}\mathsf{0} \qquad \overline{\overline{\sigma\mathbf{1}}} = \overline{\overline{\sigma}}\mathsf{1}.$$

For any $\omega \in \mathbb{O}$, let T_ω be the set of all equations of the form $\mathsf{Flipcoin}(\overline{\overline{\sigma}}) = \overline{\overline{\omega(\sigma)}}$.

▶ **Definition 15** (Provable Representability)**.** Let $f : \mathbb{O} \times \mathbb{S}^j \to \mathbb{S}$. A term $\mathsf{t} : s \Rightarrow \ldots \Rightarrow s$ of $\mathcal{POR}^\lambda$ *provably represents* f when for all strings $\sigma_1, \ldots, \sigma_j, \sigma \in \mathbb{S}$ and $\omega \in \mathbb{O}$,

$$f(\sigma_1, \ldots, \sigma_j, \omega) \quad \text{iff} \quad T_\omega \vdash_{\mathcal{POR}^\lambda} \mathsf{t}\overline{\overline{\sigma_1}} \ldots \overline{\overline{\sigma_j}} = \overline{\overline{\sigma}}.$$

▶ **Example 11.** The term $\mathsf{Flipcoin} : s \Rightarrow s$ provably represents the query function $Q(x, \omega) = \omega(x)$ of $\mathcal{POR}$, since for any $\sigma \in \mathbb{S}$ and $\omega \in \mathbb{O}$,

$$\mathsf{Flipcoin}(\overline{\overline{\sigma}}) = \overline{\overline{\omega(\sigma)}} \vdash_{\mathcal{POR}^\lambda} \mathsf{Flipcoin}(\overline{\overline{\sigma}}) = \overline{\overline{Q(\sigma, \omega)}}.$$

We consider some of the terms described above and show them to provably represent the intended functions. Let $Tail(\sigma, \omega)$ indicate the string obtained by chopping the first digit of σ, and $Trunc(\sigma_1, \sigma_2, \omega) = \sigma_1|_{\sigma_2}$.

▶ **Lemma 12.** *Terms Tail, Trunc and Cond provably represent the functions $Tail$, $Trunc$ and C, respectively.*

▶ **Theorem 13.** 1. *Any function $f \in \mathcal{POR}$ is provably represented by a term $\mathsf{t} \in \mathcal{POR}^\lambda$.*
2. *For any term $\mathsf{t} \in \mathcal{POR}^\lambda$, there is a function $f \in \mathcal{POR}$ such that f is provably repesented by t.*

Proof Sketch. 1. The proof is by induction on the structure of $f \in \mathcal{POR}$.

Base Case. Each base function is provably represented. Let us consider two examples:

- $f = E$ is provably represented by $\lambda x.\epsilon$. For any string $\sigma \in \mathbb{S}$, $\overline{\overline{E(\sigma, \omega)}} = \overline{\overline{\boldsymbol{\epsilon}}} = \epsilon$ and $\vdash_{\mathcal{POR}^\lambda} (\lambda x.\epsilon)\overline{\overline{\sigma}} = \epsilon$ is an instance of (β)-axiom. We conclude, $\vdash_{\mathcal{POR}^\lambda} (\lambda x.\epsilon)\overline{\overline{\sigma}} = \overline{\overline{E(\sigma, \omega)}}$.

- $f = Q$ is provably represented by the term $\mathsf{Flipcoin}$, as observed in Example 11 above.

Inductive Case. Each function defined by composition or bounded recursion from provably represented functions is provably represented as well. We consider bounded recursion. Let f be defined as:

$$f(\sigma_1, \dots, \sigma_n, \boldsymbol{\epsilon}, \omega) = g(\sigma_1, \dots, \sigma_n, \omega)$$
$$f(\sigma_1, \dots, \sigma_n, \sigma\mathbf{b}, \omega) = h_b(\sigma_1, \dots, \sigma_n, \sigma, f(\sigma_1, \dots, \sigma_n, \sigma, \omega), \omega)|_{k(\sigma_1, \dots, \sigma_n, \sigma)}.$$

By IH, g, h_0, h_1 and k are provably represented by the corresponding terms $\mathsf{t}_g, \mathsf{t}_{h_0}, \mathsf{t}_{h_1}, \mathsf{t}_k$. So, for any $\sigma_1, \dots, \sigma_{n+2}, \sigma \in \mathbb{S}$ and $\omega \in \mathbb{O}$:

$$T_\omega \vdash_{\mathcal{POR}^\lambda} \mathsf{t}_{\mathsf{g}} \overline{\overline{\sigma_1}} \dots \overline{\overline{\sigma_n}} = \overline{\overline{g(\sigma_1, \dots, \sigma_n, \omega)}} \qquad (\mathsf{t}_g)$$
$$T_\omega \vdash_{\mathcal{POR}^\lambda} \mathsf{t}_{h_0} \overline{\overline{\sigma_1}} \dots \overline{\overline{\sigma_{n+2}}} = \overline{\overline{h_0(\sigma_1, \dots, \sigma_{n+2}, \omega)}} \qquad (\mathsf{t}_{h_0})$$
$$T_\omega \vdash_{\mathcal{POR}^\lambda} \mathsf{t}_{h_1} \overline{\overline{\sigma_1}} \dots \overline{\overline{\sigma_{n+2}}} = \overline{h_1(\sigma_1, \dots, \sigma_{n+2}, \omega} \qquad (\mathsf{t}_{h_1})$$
$$T_\omega \vdash_{\mathcal{POR}^\lambda} \mathsf{t}_k \overline{\overline{\sigma_1}} \dots \overline{\overline{\sigma_n}} = \overline{\overline{k(\sigma_1, \dots, \sigma_n, \omega)}}. \qquad (\mathsf{t}_k)$$

We prove by induction on σ, that $T_\omega \vdash_{\mathcal{POR}^\lambda} \mathsf{t}_f \overline{\overline{\sigma_1}} \dots \overline{\overline{\sigma_n \sigma}} = \overline{\overline{f(\sigma_1, \dots, \sigma_n, \omega)}}$, where $\mathsf{t}_f = \lambda x_1 \dots \lambda x_n \lambda x.\mathsf{Rec}(\mathsf{t}_g x_1 \dots x_n, \mathsf{t}_{h_0} x_1 \dots x_n, \mathsf{t}_{h_1} x_1 \dots x_n, \mathsf{t}_k x_1 \dots x_n, x)$. Then,

- if $\sigma = \boldsymbol{\epsilon}$, then $f(\sigma_1, \dots, \sigma_n, \sigma, \omega) = g(\sigma_1, \dots, \sigma_n, \omega)$. Using the (β)-axiom we deduce, $\vdash_{\mathcal{POR}^\lambda} \mathsf{t}_f \overline{\overline{\sigma_1}} \dots \overline{\overline{\sigma_1}} \dots \overline{\overline{\sigma_n \sigma}} = \mathsf{Rec}(\mathsf{t}_g \overline{\overline{\sigma_1}} \dots \overline{\overline{\sigma_n}}, \mathsf{t}_{h_0} \overline{\overline{\sigma_1}} \dots \overline{\overline{\sigma_n}}, \mathsf{t}_{h_1} \overline{\overline{\sigma_1}} \dots \overline{\overline{\sigma_n}}, \mathsf{t}_k \overline{\overline{\sigma_1}} \dots \overline{\overline{\sigma_n}}, \overline{\overline{\sigma}})$ and

  using the axiom $\mathsf{Rec}(\mathsf{t}_g x_1 \ddots x_n, \mathsf{t}_{h_0} x_1 \dots x_n, \mathsf{t}_{h_1} x_1 \dots x_n, \mathsf{t}_k x_1 \dots x_n, \epsilon = \mathsf{t}_g x_1 \dots x_n$, we obtain $\vdash_{\mathcal{POR}^\lambda} \mathsf{t}_f \overline{\overline{\sigma_1}} \dots \overline{\overline{\sigma_n \sigma}} = \mathsf{t}_g \overline{\overline{\sigma_1}} \dots \overline{\overline{\sigma_n}}$, by (R2) and (R3). We conclude using (t_g) together with (R2).
- $\sigma = \sigma_m \mathbf{0}$, then $f(\sigma_1, \dots, \sigma_n, \sigma, \omega) = h_0(\sigma_1, \dots, \sigma_n, \sigma_m, f(\sigma_1, \dots, \sigma_n, \sigma, \omega), \omega)|_{k(\sigma_1, \dots, \sigma_n, \sigma_m)}$. By IH, suppose $T_\omega \vdash_{\mathcal{POR}^\lambda} \mathsf{t}_f \overline{\overline{\sigma_1}} \dots \overline{\overline{\sigma_n \sigma_m}} = \overline{\overline{f(\sigma_1, \dots, \sigma_n, \sigma', \omega)}}$. Thus, using the (β)-axiom $\mathsf{t}_f \overline{\overline{\sigma_1}} \dots \overline{\overline{\sigma_n \sigma}} = \mathsf{Rec}(\mathsf{t}_g \overline{\overline{\sigma}} \dots \overline{\overline{\sigma_n}}, \mathsf{t}_{h_0} \overline{\overline{\sigma_1}} \dots \overline{\overline{\sigma_n}}, \mathsf{t}_{h_1} \overline{\overline{\sigma_1}} \dots \overline{\overline{\sigma_n}}, \mathsf{t}_k \overline{\overline{\sigma_1}} \dots \overline{\overline{\sigma_n}}, \overline{\overline{\sigma}}$ the axiom $\mathsf{Rec}(g, h_0, h_1, k, x\mathsf{0}) = \mathsf{Trunc}(\mathsf{t}_{h_0} x(\mathsf{Rec}(g, h_0, h_1, k, \mathsf{0})), kx)$ and IH we deduce, $\vdash_{\mathcal{POR}^\lambda} \mathsf{t}_f \overline{\overline{\sigma_1}} \dots \overline{\overline{\sigma_n \sigma}} = \mathsf{Trunc}(\mathsf{t}_{h_0} \overline{\overline{\sigma_1}} \dots \overline{\overline{\sigma_n \sigma_m}} \overline{\overline{f(\sigma_1, \dots, \sigma_n, \sigma_m, \omega)}}, \mathsf{t}_k \overline{\overline{\sigma_1}} \dots \overline{\overline{\sigma_n}})$, by (R2) and (R3). Using (t_{h_0}) and (t_k) we conclude using (R3) and (R2): $\vdash_{\mathcal{POR}^\lambda} \mathsf{t} \overline{\overline{\sigma_1}} \dots \overline{\overline{\sigma_n \sigma}} = \overline{\overline{h_0(\sigma_1, \dots, \sigma_n, \sigma_m,}}$ $\overline{\overline{f(\sigma_1, \dots, \sigma, \sigma_m, \omega)|_{k(\sigma_1, \dots \sigma_n, \sigma_m)}}}$.
- the case $\sigma = \sigma_m \mathbf{1}$ is proved in a similar way.

2. It is a consequence of the normalization property for the simply typed λ-calculus: a β-normal term $\mathsf{t} : s \Rightarrow \dots \Rightarrow s$ cannot contain variables of higher types. By exhaustively inspecting possible normal forms, representability is checked. ◀

▶ **Corollary 2.** For any function $f : \mathbb{S}^j \times \mathbb{O} \to \mathbb{S}$, $f \in \mathcal{POR}$ when f is provably represented by some term $\mathsf{t} : s \Rightarrow \dots \Rightarrow s \in \mathcal{POR}^\lambda$.

The Theory $I\mathcal{POR}^\lambda$. We introduce a first-order *intuitionistic* theory $I\mathcal{POR}^\lambda$, which extends $\mathcal{POR}^\lambda$ with basic predicate calculus and a restricted induction principle. We also define $IR\Sigma_1^b$-NIA as a variant of $R\Sigma_1^b$-NIA having the intuitionistic predicate calculus as its logical basis. All theorems of $\mathcal{POR}^\lambda$ and $IR\Sigma_1^b$-NIA are provable in $I\mathcal{POR}^\lambda$. In fact, $I\mathcal{POR}^\lambda$ can be seen as an extension of $\mathcal{POR}^\lambda$ and provides a language to associate derivations in $IR\Sigma_1^b$-NIA with poly-time computable functions, corresponding to terms of $I\mathcal{POR}^\lambda$.

The language of $I\mathcal{POR}^\lambda$ extends that of $\mathcal{POR}^\lambda$ with (a translation for) all expressions of $R\Sigma_1^b$-NIA. In particular, the grammar for terms of $I\mathcal{POR}^\lambda$ is precisely the same as that of Definition 12, while that for formulas is defined below.

▶ **Definition 16.** *Formulas of* $\mathcal{IPOR}^\lambda$ are defined as follows: i. all equations of $\mathcal{POR}^\lambda$ $\mathtt{t} = \mathtt{u}$, are formulas of $\mathcal{IPOR}^\lambda$; ii. for any (possibly open) $\mathcal{POR}^\lambda$-term $\mathtt{t}, \mathtt{u}$, $\mathtt{t} \subseteq \mathtt{u}$ and $\mathtt{Flip}(\mathtt{t})$ are formulas of $\mathcal{IPOR}^\lambda$; iii. formulas of $\mathcal{IPOR}^\lambda$ are closed under $\wedge, \vee, \rightarrow, \forall, \exists$.

We adopt the standard conventions: $\bot := \mathsf{0} = \mathsf{1}$ and $\neg F := F \rightarrow \bot$. The notions of Σ^b_0- and Σ^b_1-formula of $\mathcal{IPOR}^\lambda$ are precisely as those for $R\Sigma^b_1$-NIA.

▶ Remark 14. Any formula of $R\Sigma^b_1$-NIA can be seen as a formula of $\mathcal{IPOR}^\lambda$, where each occurrence of $\mathtt{0}$ is replaced by $\mathsf{0}$, of $\mathtt{1}$ by $\mathsf{1}$, of $\frown$ by $\circ$ (usually omitted), of $\times$ by Times. In the following, we assume that any formula of $R\Sigma^b_1$-NIA is a formula of $\mathcal{IPOR}^\lambda$, modulo the substitutions defined above.

▶ **Definition 17.** The axioms of $\mathcal{IPOR}^\lambda$ include standard rules of the intuitionistic first-order predicate calculus, usual rules for the equality symbol, plus the following axioms: 1. all axioms of $\mathcal{POR}^\lambda$, 2. $x \subseteq y \leftrightarrow \mathsf{Sub}(x, y) = \mathsf{1}$, 3. $x = \epsilon \vee x = \mathsf{Tail}(x)\mathsf{0} \vee x = \mathsf{Tail}(x)\mathsf{1}$, 4. $\mathsf{0} = \mathsf{1} \rightarrow x = \epsilon$, 5. $\mathsf{Cond}(x, y, z, w) = w' \leftrightarrow (x = \epsilon \wedge w' = y) \vee (x = \mathsf{Tail}(x)\mathsf{0} \wedge w' = z) \vee (x = \mathsf{Tail}(x)\mathsf{1} \wedge w' = w)$, 6. $\mathtt{Flip}(x) \leftrightarrow \mathsf{Flipcoin}(x) = \mathsf{1}$, 7. any formula of the form:

$$(F(\epsilon) \wedge \forall x.(F(x) \rightarrow F(x\mathsf{0})) \wedge \forall x.(F(x) \rightarrow F(x\mathsf{1}))) \rightarrow \forall y.F(y),$$

where F is of the form $\exists z \preceq \mathtt{t}.\mathtt{u} = \mathtt{v}$, with $\mathtt{t}$ containing only first-order open variables.

▶ **Notation 2.** We refer to a formula of the form $\exists z \preceq \mathtt{t}.\mathtt{u} = \mathtt{v}$, with $\mathtt{t}$ containing only first-order open variables, as an **NP**-*predicate.*

Now that $\mathcal{IPOR}^\lambda$ has been introduced we show that all theorems of both $\mathcal{POR}^\lambda$ and the intuitionistic version of $R\Sigma^b_1$-NIA are derived in it. First, Proposition 3 is established inspecting all rules of $\mathcal{POR}^\lambda$.

▶ **Proposition 3.** Any theorem of $\mathcal{POR}^\lambda$ is a theorem of $\mathcal{IPOR}^\lambda$.

Then, we show that every theorem of $IR\Sigma^b_1$-NIA is derivable in $\mathcal{IPOR}^\lambda$. To do so, we prove a few properties concerning $\mathcal{IPOR}^\lambda$. In particular, its recursion schema differs from that of $IR\Sigma^b_1$-NIA as dealing with formulas of the form $\exists y \preceq \mathtt{t}.\mathtt{u} = \mathtt{v}$ and not with all the Σ^b_1-ones. The two schemas are related by Proposition 4 proved by induction on the structure of formulas.

▶ **Proposition 4.** For any Σ^b_0-formula $F(x_1, \dots, x_n)$ in $\mathcal{RL}$, there is a term $\mathtt{t}_F(x_1, \dots, x_n)$ of $\mathcal{POR}^\lambda$ such that: 1. $\vdash_{\mathcal{IPOR}^\lambda} F \leftrightarrow \mathtt{t}_F = \mathsf{0}$, 2. $\vdash_{\mathcal{IPOR}^\lambda} \mathtt{t}_F = \mathsf{0} \vee \mathtt{t}_F = \mathsf{1}$.

This leads us to the following corollary and to Theorem 15, realting $\mathcal{IPOR}^\lambda$ and $IR\Sigma^b_1$-NIA.

▶ **Corollary 3.** i. For any Σ^b_0-formula F, $\vdash_{\mathcal{IPOR}^\lambda} F \vee \neg F$; ii. For any closed Σ^b_0-formula of $\mathcal{RL}$ F and $\omega \in \mathbb{O}$, either $T_\omega \vdash_{\mathcal{IPOR}^\lambda} F$ or $T_\omega \vdash_{\mathcal{IPOR}^\lambda} \neg F$.

▶ **Theorem 15.** *Any theorem of* $IR\Sigma^b_1$*-NIA is a theorem of* $\mathcal{IPOR}^\lambda$*.*

Proof. First, observe that, as a consequence of Proposition 4, for any $\Sigma^b - 1$-formula $F = \exists x_1 \preceq t_1. \dots \exists x_n \preceq t_n.G$ in $\mathcal{RL}$, $\vdash_{\mathcal{IPOR}^\lambda} F \leftrightarrow \exists x_1 \preceq \mathtt{t}_1. \dots \exists x_n \preceq \mathtt{t}_n.\mathtt{t}_G = \mathsf{0}$, any instance of the Σ^b_1-recursion schema of $IR\Sigma^b_1$-NIA is derivable in $\mathcal{IPOR}^\lambda$ from the **NP**-induction schema. Then, we conclude noticing that basic axioms of $IR\Sigma^b_1$-NIA are provable in $\mathcal{IPOR}^\lambda$. ◀

▶ **Corollary 4.** For any closed Σ^b_0-formula F and $\omega \in \mathbb{O}$, either $T_\omega \vdash_{\mathcal{IPOR}^\lambda} F$ or $T_\omega \vdash_{\mathcal{IPOR}^\lambda}$ $\neg F$.

Due to Corollary 4 we establish the following Lemma 16.

▶ **Lemma 16.** *For any closed Σ^b_0-formula F and $\omega \in \mathbb{O}$, either $T_\omega \vdash_{\mathcal{IPOR}^\lambda} F$ iff $\omega \in [\![F]\!]$.*

Proof. ($\Rightarrow$) By induction on the structure of rules for $\mathcal{IPOR}$. ($\Leftarrow$) For Corollary 4, either $T_\omega \vdash_{\mathcal{IPOR}^\lambda} F$ or $T_\omega \vdash_{\mathcal{IPOR}^\lambda} \neg F$. Hence, if $\omega \in [\![F]\!]$, then it cannot be $T_\omega \vdash_{\mathcal{IPOR}^\lambda} \neg F$ (by soundness). We conclude $T_\omega \vdash_{\mathcal{IPOR}^\lambda} F$. ◀

Realizability. We introduce realizability internal to $\mathcal{IPOR}^\lambda$. As a corollary, we obtain that from any derivation in $IR\Sigma^b_1$-NIA – actually, in $\mathcal{IPOR}^\lambda$ – of a formula in the form $\forall x.\exists y.F(x,y)$, one can extract a functional term of $\mathcal{POR}^\lambda$ $\mathsf{f} : s \Rightarrow s$, such that $\vdash_{\mathcal{IPOR}^\lambda} \forall x.F(x, \mathsf{f}x)$. This allows us to conclude that if f is Σ^b_1-representable in $IR\Sigma^b_1$-NIA, then $f \in \mathcal{POR}$.

▶ **Notation 3.** Let $\mathbf{x}, \mathbf{y}$ denote finite sequences of term variables, (resp.) $x_1, \dots, x_n$ and $y_1, \dots, y_k$ and $\mathbf{x}(\mathbf{y})$ be an abbreviation for $y_1(\mathbf{x}), \dots, y_k(\mathbf{x})$. Let Λ be a shorthand for the empty sequence and $y(\Lambda) := y$.

▶ **Definition 18.** *Formulas $x \circledR F$ are defined by induction as follows:*

$$\begin{aligned}
\Lambda \circledR F &:= F \\
\mathbf{x}, \mathbf{y} \circledR (G \wedge H) &:= (\mathbf{x} \circledR G) \wedge (\mathbf{y} \circledR H) \\
z, \mathbf{x}, \mathbf{y} \circledR (G \vee H) &:= (z = \mathsf{0} \wedge \mathbf{x} \circledR G) \vee (z \neq \mathsf{0} \wedge \mathbf{y} \circledR H) \\
\mathbf{y} \circledR (G \to H) &:= \forall \mathbf{x}.(\mathbf{x} \circledR G \to \mathbf{y}(\mathbf{x}) \circledR H) \wedge (G \to H) \\
z, \mathbf{x} \circledR \exists y.G &:= \mathbf{x} \circledR G\{z/y\} \\
\mathbf{x} \circledR \forall y.G &:= \forall y.(\mathbf{x}(y) \circledR G),
\end{aligned}$$

where no variable in $\mathbf{x}$ is free in F. Given terms $\mathbf{t} = \mathsf{t}_1, \dots, \mathsf{t}_n$ we let $\mathbf{t} \circledR F := (\mathbf{x} \circledR F)\{\mathbf{t}/\mathbf{x}\}$.

We relate the derivability of these new formulas with that of formulas of $\mathcal{IPOR}^\lambda$. Proofs below are by induction (resp.) on the structure of $\mathcal{IPOR}^\lambda$-formulas and on the height of derivations.

▶ **Theorem 17** (Soundness). *If $\vdash_{\mathcal{IPOR}^\lambda} \mathbf{t} \circledR F$, then $\vdash_{\mathcal{IPOR}^\lambda} F$.*

▶ **Notation 4.** Given $\Gamma = F_1, \dots, F_n$, let $\mathbf{x} \circledR \Gamma$ be a shorthand for $\mathbf{x}_1 \circledR F_1, \dots, \mathbf{x}_n \circledR F_n$.

▶ **Theorem 18** (Completeness). *If $\vdash_{\mathcal{IPOR}^\lambda} F$, then $\mathbf{t}$ such that $\vdash_{\mathcal{IPOR}^\lambda} \mathbf{t} \circledR F$.*

Proof. We prove that if $\Gamma \vdash_{\mathcal{IPOR}^\lambda} F$, there exist terms $\mathbf{t}$ such that $\mathbf{x} \circledR \Gamma \vdash_{\mathcal{IPOR}^\lambda} \mathbf{t}\mathbf{x}_1 \dots, \mathbf{x}_n \circledR F$. The proof is by induction on the derivation of $\Gamma \vdash_{\mathcal{IPOR}^\lambda} F$. Let us consider just one example:

$$\dfrac{\begin{array}{c}\vdots \\ \Gamma \vdash G\end{array}}{\Gamma \vdash G \vee H}\ \vee R_1$$

By IH, there exist terms $\mathbf{u}$, such that $\mathbf{t} \circledR \Gamma \vdash_{\mathcal{IPOR}^\lambda} \mathbf{tu} \circledR G$. Since $x, y \circledR G \vee H$ is defined as $(x = \mathsf{0} \wedge y \circledR G) \vee (x \neq \mathsf{0} \wedge y \circledR H)$, we can take $\mathbf{t} = \mathsf{0}, \mathbf{u}$. ◀

▶ **Corollary 5.** Let $\forall x.\exists y.F(x,y)$ be a closed term of $\mathcal{IPOR}^\lambda$, where F is a Σ^b_1-formula. Then, there is a closed term $\mathsf{t} : s \Rightarrow s$ of $\mathcal{POR}^\lambda$ such that $\vdash_{\mathcal{IPOR}^\lambda} \forall x.F(x, \mathsf{t}x)$.

Proof. By Theorem 18, there exist $\mathbf{t} = \mathsf{t}, w$ such that $\vdash_{\mathcal{IPOR}^\lambda} \mathbf{t} \circledR \forall x.\exists y.F(x,y)$. So, $\mathbf{t} \circledR \forall x.\exists y.F(x,y) \equiv \forall x.(\mathbf{t}(x) \circledR \exists y.F(x,y)) \equiv \forall x.(w(x) \circledR F(x,\mathsf{t}x))$. From this, by Theorem 17, we deduce $\vdash_{\mathcal{IPOR}^\lambda} \forall x.F(x,\mathsf{t}x)$. ◀

Now, we have all the ingredients to prove that if a function is Σ^b_1-representable in $IR\Sigma^b_1$-NIA, then it is in $\mathcal{POR}$.

▶ **Corollary 6.** For any function $f : \mathbb{O} \times \mathbb{S} \to \mathbb{S}$, if there is a closed Σ^b_1-formula in $\mathcal{RL}$ $F(x,y)$, such taht:

1. $IR\Sigma^b_1\text{-NIA} \vdash \forall x.\exists! y.F(x,y)$
2. $[\![F(\overline{\overline{\sigma_1}}, \overline{\overline{\sigma_2}})]\!] = \{\omega \mid f(\sigma_1,\omega) = \sigma_2\}$,

then $f \in \mathcal{POR}$.

Proof. Since $\vdash_{IR\Sigma^b_1\text{-NIA}} \forall x.\exists! y.F(x,y)$, by Theorem 15 $\vdash_{\mathcal{IPOR}^\lambda} \forall x.\exists! y.F(x,y)$. Then, from $\vdash_{\mathcal{IPOR}^\lambda} \forall x.\exists y.F(x,y)$, we deduce $\vdash_{\mathcal{IPOR}^\lambda} \forall x.F(x,\mathsf{g}x)$ for some closed term $\mathsf{g} \in \mathcal{POR}^\lambda$, by Corollary 6. Furthermore, by Theorem 13.2, there is a $g \in \mathcal{POR}$ such that for any $\sigma_1,\sigma_2 \in \mathbb{S}$ and $\omega \in \mathbb{O}$, $g(\sigma_1,\omega) = \sigma_2$, when $T_\omega \vdash_{\mathcal{IPOR}^\lambda} \mathsf{g}\overline{\overline{\sigma_1}} = \overline{\overline{\sigma_2}}$. So, by Proposition 3, for any $\sigma_1,\sigma_2 \in \mathbb{S}$ and $\omega \in \mathbb{O}$ if $g(\sigma_1,\omega) = \sigma_2$, then $T_\omega \vdash_{\mathcal{IPOR}^\lambda} \mathsf{g}\overline{\overline{\sigma_1}} = \overline{\overline{\sigma_2}}$ and so $T_\omega \vdash_{\mathcal{IPOR}^\lambda} F(\overline{\overline{\sigma_1}}, \overline{\overline{\sigma_2}})$. By Lemma 16, $T_\omega \vdash_{\mathcal{IPOR}^\lambda} F(\overline{\overline{\sigma_1}}, \overline{\overline{\sigma_2}})$, when $\omega \in [\![F(\overline{\overline{\sigma_1}}, \overline{\overline{\sigma_2}})]\!]$, that is $f(\sigma_1,\omega) = \sigma_2$. But then $f = g$, so since $g \in \mathcal{POR}$ also $f \in \mathcal{POR}$. ◀

$\forall$**NP**-*Conservativity of* $\mathcal{IPOR}^\lambda + EM$ *over* $\mathcal{IPOR}^\lambda$. Corollary 6 is already close to the result we are looking for. The remaining step to conclude our proof is its extension from intuitionistic $IR\Sigma^b_1$-NIA to classical $R\Sigma^b_1$-NIA, showing that any function which is Σ^b_1-representable in $R\Sigma^b_1$-NIA is also in $\mathcal{POR}$. The proof adapts method by [12]. We start by considering an extension of $\mathcal{IPOR}^\lambda$ via EM and show that the realizability interpretation extends to it so that for any of its closed theorems $\forall x.\exists y \preceq \mathsf{t}.F(x,y)$, being F a Σ^b_1-formula, there is a closed term $\mathsf{t} : s \Rightarrow s$ of $\mathcal{POR}^\lambda$ such that $\vdash_{\mathcal{IPOR}^\lambda} \forall x.F(x,\mathsf{t}x)$.

Let EM be the excluded-middle schema $F \vee \neg F$, and Markov's principle be defined as follows $\neg\neg(\exists x.F \to (existsx)F)$ where F is a Σ^b_1-formula.

▶ **Proposition 5.** For any Σ^b_1-formula F, if $\vdash_{\mathcal{IPOR}^\lambda + EM} F$, then $\vdash_{\mathcal{IPOR}^\lambda + (Markov)} F$.

Proof Sketch. The proof is by double negation translation with the following two remarks: 1. for any Σ^b_0-formula F, $\vdash_{\mathcal{IPOR}^\lambda} \neg\neg F \to F$; 2. using (Markov), the double negation of an instance of the **NP**-induction can be shown equivalent the **NP**-induction schema. ◀

We conclude by that the realizability interpretation defined above extends to $\mathcal{IPOR}^\lambda$+(Markov), that is for any closed theorem $\forall x.\exists y \preceq \mathsf{t}.F(x,y)$ with F Σ^b_1-formula of $\mathcal{IPOR}^\lambda$+(Markov) there is a closed term of $\mathcal{POR}^\lambda$ $\mathsf{t} : s \Rightarrow s$ such that $\vdash_{\mathcal{IPOR}^\lambda} \forall x.F(x,\mathsf{t}x)$.

Let assume given an encoding $\sharp : (s \Rightarrow s) \Rightarrow s$ in $\mathcal{IPOR}^\lambda$ of first-order unary functions as strings, together with a "decoding" function $\mathsf{app} : s \Rightarrow s \Rightarrow s$ satisfying $\vdash_{\mathcal{IPOR}^\lambda} \mathsf{app}(\sharp\mathsf{f}, x) = \mathsf{f}x$. Moreover, let $x * y := \sharp(\lambda z.\mathsf{BAnd}(\mathsf{app}(x,z), \mathsf{app}(y,z)))$ and $T(x) := \exists y.(\mathsf{B}(\mathsf{app}(x,y)) = 0)$. There is a *meet semi-lattice* structure on the set of terms of type s defined by $\mathsf{t} \sqsubseteq \mathsf{u}$ when $\vdash_{\mathcal{IPOR}^\lambda} T(\mathsf{u}) \to T(\mathsf{t})$ with top element $\underline{\mathbf{1}} := \sharp(\lambda x.1)$ and meet given by $x * y$. Indeed, from $T(x*1) \leftrightarrow T(x)$, $x \sqsubseteq \underline{\mathbf{1}}$ follows, Moreover, from $\mathsf{B}(\mathsf{app}(x,\mathsf{u}) = 0$, we obtain $\mathsf{B}(\mathsf{app}(x*y,\mathsf{u})) = \mathsf{BAnd}(\mathsf{app}(x,\mathsf{u}), \mathsf{app}(y,\mathsf{u})) = 0$, whence $T(x) \to T(x*y)$, i.e. $x * y \sqsubseteq x$. One can similarly prove $x * y \sqsubseteq y$. Finally, from $T(x) \to T(v)$ and $T(y) \to T(v)$, we deduce $T(x*y) \to T(v)$, by observing that $\vdash_{\mathcal{IPOR}^\lambda} T(x*y) \to T(y)$. Notice that the formula $T(x)$ is *not* a Σ^b-one, as its existential quantifier is not bounded.

▶ **Definition 19.** For any $\mathcal{IPOR}^\lambda$-formula F and fresh variable x, we define formulas $x \Vdash F$:

$$
\begin{aligned}
x \Vdash F &:= F \vee T(x) && (F \text{ atomic})\\
x \Vdash G \wedge H &:= x \Vdash G \wedge x \Vdash H\\
x \Vdash G \vee H &:= x \Vdash G \vee x \Vdash H\\
x \Vdash G \to H &:= \forall y.(y \Vdash G \to x * y \Vdash H)\\
x \Vdash \exists y.G &:= \exists y.x \Vdash G\\
x \Vdash \forall y.G &:= \forall y.x \Vdash G.
\end{aligned}
$$

▶ **Lemma 19.** *If F is provable in $\mathcal{IPOR}^\lambda$ without using* **NP***-induction, then $x \Vdash F$ is provable in $\mathcal{IPOR}^\lambda$.*

Proof Sketch. By induction on the structure of formulas of $\mathcal{IPOR}^\lambda$ as in [14]. ◀

▶ **Lemma 20.** *Let $F = \exists x \preceq \mathsf{t}.G$, where F is a Σ^b_0-formula. Then, there is a term $\mathsf{u}_F : s$ with $FV(\mathsf{u}_F) = FV(G)$ such that $\vdash_{\mathcal{IPOR}^\lambda} F \leftrightarrow T(\mathsf{u}_F)$.*

Proof. Since $G(x)$ is a Σ^b_0-formula, for all terms $\mathsf{u} : s$, $\vdash_{\mathcal{IPOR}^\lambda} G(x) \leftrightarrow \mathsf{u}_{x\preceq\mathsf{t}\wedge G}(x) = 0$, where $\mathsf{t}_{x\preceq\mathsf{t}\wedge G}$ has the free variables of t and G. Let $H(x)$ be a Σ^b_0-formula, it is shown by induction on its structure that for any term $\mathsf{v} : s$, $\mathsf{t}_{H(\mathsf{v})} = \mathsf{t}_H(\mathsf{v})$. Then, $\vdash_{\mathcal{IPOR}^\lambda}\vdash F \leftrightarrow \exists x.\mathsf{t}_{x\preceq\mathsf{u}\wedge G}(x) = 0 \leftrightarrow \exists x.T(\sharp(\lambda x.\mathsf{t}_{x\preceq\mathsf{t}\wedge G}(x)))$. So, we let $\mathsf{u}_F = \sharp(\lambda x.\mathsf{t}_{x\preceq\mathsf{u}\wedge G}(x))$. ◀

From which we deduce the following three properties: i. $\vdash_{\mathcal{IPOR}^\lambda} (x \Vdash F) \leftrightarrow (F \vee T(x))$; ii. $\vdash_{\mathcal{IPOR}^\lambda} (x \Vdash F) \leftrightarrow (F \to T(x))$; iii. $\vdash_{\mathcal{IPOR}^\lambda} (x \Vdash \neg\neg F) \leftrightarrow (F \vee T(x))$, where F is a Σ^b_1-formula.

▶ **Corollary 7** (Markov's Principle)**.** If F is a Σ^b_1-formula, then $\vdash_{\mathcal{IPOR}^\lambda} x \Vdash \neg\neg F \to F$.

To define the extension $(\mathcal{IPOR}^\lambda)^*$ of $\mathcal{IPOR}^\lambda$, we introduce PIND(F) as:

$$(F(\epsilon) \wedge (\forall x.(F(x) \to F(x0)) \wedge \forall x.(F(x) \to F(x1)))) \to \forall x.F(x).$$

Observe that if $F(x)$ is a formula of the form $\exists y \preceq \mathsf{t}.\mathsf{u} = \mathsf{v}$, then $z \Vdash \text{PIND}(F)$ is of the form PIND$(F(x) \vee T(z))$, which is *not* an instance of the **NP**-induction schema.

▶ **Definition 20.** Let $(\mathcal{IPOR}^\lambda)^*$ be the theory extending $\mathcal{POR}^\lambda$ with all instances of the induction schema PIND$(F(x) \vee G)$, where $F(x)$ is of the form $\exists y \preceq \mathsf{t}.\mathsf{u} = \mathsf{v}$, and G is an arbitrary formula with $x \notin FV(G)$.

The following Proposition relates derivability in $\mathcal{IPOR}^\lambda$ and in $(\mathcal{IPOR}^\lambda)^*$.

▶ **Proposition 6.** For any Σ^b_1-formula F, if $\vdash_{\mathcal{IPOR}^\lambda} F$, then $\vdash_{(\mathcal{IPOR}^\lambda)^*} x \Vdash F$.

Finally, we extend realizability to $(\mathcal{IPOR}^\lambda)^*$ by constructing a realizer fo PIND$(F(x) \vee G)$.

▶ **Lemma 21.** *Let $F(x) : \exists y \preceq t.u = 0$ and G be any formula not containing free occurrences of x. Then, there exist terms t such that $\vdash_{\mathcal{IPOR}^\lambda} \mathsf{t}$ ® PIND$(F(x) \vee G)$.*

So, by Theorem 17, for any Σ^b_1-formula F and formula G, with $x \notin FV(F)$, $\vdash_{\mathcal{IPOR}^\lambda}$ PIND$(F(x) \vee G)$.

▶ **Corollary 8** ($\forall$**NP**-Conservativity)**.** Let F be a Σ^b_1-fromula. If $\vdash_{\mathcal{IPOR}^\lambda+EM} \forall x.\exists y \preceq \mathsf{t}.F(x,y)$, then $\vdash_{\mathcal{IPOR}^\lambda} \forall x.\exists y \preceq \mathsf{t}.F(x,y)$.

We conclude the proof establishing the following Proposition 7.

▶ **Proposition 7.** Let $\forall x.\exists y \preceq \mathsf{t}.F(x,y)$ be a closed term of $\mathcal{IPOR}^{\lambda}$+(Markov), where F is a Σ^b_1-formula. Then, there is a closed term of $\mathcal{POR}^{\lambda}$ $\mathsf{t} : s \Rightarrow s$ such that $\vdash_{\mathcal{IPOR}^{\lambda}} \forall x.F(x, \mathsf{t}x)$.

Proof. If $\vdash_{\mathcal{IPOR}^{\lambda}+(Markov)} \forall x.\exists y.F(x,y)$, then by Proposition 2, also $\vdash_{\mathcal{IPOR}^{\lambda}+(Markov)} \exists y \preceq \mathsf{t}.F(x,y)$. Moreover, $\vdash_{(\mathcal{IPOR}^{\lambda})^*} z \Vdash \exists y \preceq \mathsf{t}.F(x,y)$. Then, let us consider $G = \exists y \preceq \mathsf{t}.F(x,y)$. Taking $\mathsf{v} = \mathsf{u}_G$, by Lemma 20, we deduce $\vdash_{(\mathcal{IPOR}^{\lambda})^*} G$ and, thus, by Lemma 19 and 21, we conclude that there exist $\mathbf{t}, \mathbf{u}$ such that $\vdash_{\mathcal{IPOR}^{\lambda}} \mathbf{t}, \mathbf{u} \circledR G$, which implies $\vdash_{\mathcal{IPOR}^{\lambda}} F(x, \mathbf{t}x)$ and so $\vdash_{\mathcal{IPOR}^{\lambda}} \forall x.(F(x), \mathbf{t}x)$. ◀

So, by Proposition 5, if $\vdash_{\mathcal{IPOR}^{\lambda}+EM} \forall x.\exists y \preceq \mathsf{t}.F(x,y)$, being F a closed Σ^b_1-formula, then there is a closed term of $\mathcal{POR}$ $\mathsf{t} : s \Rightarrow s$ such that $\vdash_{\mathcal{IPOR}^{\lambda}} \forall x.F(x, \mathsf{t}x)$. Finally, we conclude the desired Corollary 9.

▶ **Corollary 9.** Let $R\Sigma^b_1$-NIA $\vdash \forall x.\exists y \preceq t.F(x,y)$, where F is a Σ^b_1-formula with only x, y free. For any $f : \mathbb{S} \times \mathbb{O} \to \mathbb{S}$, if $\forall x.\exists y \preceq t.F(x,y)$ represents f so that:

1. $R\Sigma^b_1$-NIA $\vdash \forall x.\exists! y.F(x,y)$
2. $[\![F(\overline{\overline{\sigma_1}}, \overline{\overline{\sigma_2}})]\!] = \{\omega \mid f(\sigma_1, \omega) = \sigma_2\}$,

then $f \in \mathcal{POR}$.

B Proofs from Section 3

B.1 From RFP to $\mathcal{POR}$

The goal of this section is to establish a correspondence between **RFP** and $\mathcal{POR}$. This passes through Proposition 8, which assesses the equivalence between **RFP** and the intermediate class **SFP** and Proposition 9, which concludes the proof showing the equivalence between $\mathcal{POR}$ and **SFP**. To establish rigorously the results mentioned above, we fix some definitions. We start with those of STMs and their configurations:

▶ **Definition 21** (Stream Turing Machine). A *stream Turing machine* is a quadruple $M := \langle \mathcal{Q}, q_0, \Sigma, \delta \rangle$, where:

- $\mathcal{Q}$ is a finite set of states ranged over by q_i and similar meta-variables;
- $q_0 \in \mathcal{Q}$ is an initial state;
- Σ is a finite set of characters ranged over by c_i *et simila*;
- $\delta : \hat{\Sigma} \times \mathcal{Q} \times \hat{\Sigma} \times \mathbb{B} \longrightarrow \hat{\Sigma} \times \mathcal{Q} \times \hat{\Sigma} \times \{L, R\}$ is a transition function describing the new configuration reached by the machine.

L and R are two fixed and distinct symbols, e.g. $\mathbf{0}$ and $\mathbf{1}$, $\hat{\Sigma} = \Sigma \cup \{\circledast\}$ and $\circledast$ represents the *blank character*, such that $\circledast \notin \Sigma$. Without loss of generality, in the following, we will use STMs with $\Sigma = \mathbb{B}$.

▶ **Definition 22** (Configuration of STM). The *configuration of an STM* is a quadruple $\langle \sigma, q, \tau, \eta \rangle$, where:

- $\sigma \in \{\mathbf{0}, \mathbf{1}, \circledast\}^*$ is the portion of the work tape on the left of the head;
- $q \in \mathcal{Q}$ is the current state of the machine;
- $\tau \in \{\mathbf{0}, \mathbf{1}, \circledast\}^*$ is the portion of the work tape on the right of the head;
- $\eta \in \mathbb{B}^{\mathbb{N}}$ is the portion of the oracle tape that has not been read yet.

Thus, we give the definition of the family of reachability relations for STM machines.

▶ **Definition 23** (Stream Machine Reachability Functions)**.** Given an STM $\mathcal{S}$ with transition function δ, we denote with $\vdash_\delta$ its standard step function and we call $\{\triangleright^n_{\mathcal{S}}\}_n$ the smallest family of relations for which:

$$\langle\sigma, q, \tau, \eta\rangle \triangleright^0_M \langle\sigma, q, \tau, \eta\rangle$$

$$\Big(\langle\sigma, q, \tau, \eta\rangle \triangleright^n_M \langle\sigma', q', \tau', \eta'\rangle\Big) \wedge \Big(\langle\sigma', q', \tau', \eta'\rangle \vdash_\delta \langle\sigma'', q', \tau'', \eta''\rangle\Big) \rightarrow \Big(\langle\sigma, q, \tau, \eta\rangle \triangleright^{n+1}_M \langle\sigma'', q'\tau'', \eta''\rangle\Big)$$

▶ **Definition 24** (STM Computation)**.** Given an STM, $\mathcal{S} = \langle\mathcal{Q}, q_0, \Sigma, \delta\rangle$, $\eta : \mathbb{N} \longrightarrow \mathbb{B}$ and a function $g : \mathbb{N} \longrightarrow \mathbb{B}$, we say that $\mathcal{S}$ *computes* g, written $f_{\mathcal{S}} = g$ iff for every string $\sigma \in \mathbb{S}$, and oracle tape $\eta \in \mathbb{B}^{\mathbb{N}}$, there are $n \in \mathbb{N}$, $\tau \in \mathbb{S}, q' \in \mathcal{Q}$, and a function $\psi : \mathbb{N} \longrightarrow \mathbb{B}$ such that:

$$\langle\boldsymbol{\epsilon}, q_0, \sigma, \eta\rangle \ \triangleright^n_{\mathcal{S}} \ \langle\gamma, q', \tau, \psi\rangle,$$

and $\langle\gamma, q', \tau, \psi\rangle$ is a final configuration for $\mathcal{S}$ with $f_{\mathcal{S}}(\sigma, \eta)$ being the longest suffix of γ not including $\circledast$.

Similar notations are employed for all the families of Turing-like machines we define in this paper. However, PTMs are an exception since they compute distributions over $\mathbb{S}$ instead of functions $\mathbb{S} \times \mathbb{O} \longrightarrow \mathbb{S}$.

▶ **Definition 25** (Probabilistic Turing Machines)**.** A Probabilistic Turing Machine (PTM) is a TM with two transition functions $\delta_{\mathbf{0}}$ and $\delta_{\mathbf{1}}$ at each step of the computation, $\delta_{\mathbf{0}}$ is applied with probability $\frac{1}{2}$, otherwise $\delta_{\mathbf{1}}$ is applied. Given a PTM $\mathcal{M}$, a configuration $\langle\sigma, q, \tau\rangle$, we define its semantics on configurations $\langle\sigma, q, \tau\rangle$ as the following *sequence of random variables*:

$$\forall\eta \in \mathbb{B}^{\mathbb{N}}.X^{\langle\sigma,q,\tau\rangle}_{M,0} := \eta \mapsto \langle\sigma, q, \tau\rangle$$

$$\forall\eta \in \mathbb{B}^{\mathbb{N}}.X^{\langle\sigma,q,\tau\rangle}_{M,n+1} := \eta \mapsto \begin{cases} \delta_{\mathbf{b}}(X^{\langle\sigma,q,\tau\rangle}_{M,n}(\eta)) & \text{if } \eta(n) = \mathbf{b} \wedge \exists\langle\sigma', q'\tau'\rangle.\delta_{\mathbf{b}}(X^{\langle\sigma,q,\tau\rangle}_{M,n}(\eta)) = \langle\sigma', q', \tau'\rangle \\ X^{\langle\sigma,q,\tau\rangle}_{M,n}(\eta) & \text{if } \eta(n) = \mathbf{b} \wedge \neg\exists\langle\sigma', q'\tau'\rangle.\delta_{\mathbf{b}}(X^{\langle\sigma,q,\tau\rangle}_{M,n}(\eta)) = \langle\sigma', q', \tau'\rangle \end{cases}$$

Intuitively, the variable $X^{\langle\sigma,q,\tau\rangle}_{\mathcal{M},n}$ maps each possible cylinder $\eta : \mathbb{N} \longrightarrow \mathbb{B}$ to the configuration reached by the machine after exactly n transitions where the first transition step has employed $\delta_{\eta(0)}$, the second has employed $\delta_{\eta(1)}$ and so on. We say that a PTM $\mathcal{M}$ computes $Y_{\mathcal{M},\sigma}$ iff $\exists t \in \mathbb{N}.\forall\sigma.X^{\langle\sigma,q_0,\tau\rangle}_{\mathcal{M},t}$ is final. In such case, for every η, $Y_{\mathcal{M},\sigma}(\eta)$ is the longest suffix of the leftmost element in $X^{\langle\sigma,q_0,\epsilon\rangle}_{\mathcal{M},t}(\eta)$, which does not contain $\circledast$.

We start with the proof of the equivalence between the class of the PTM's polytime functions and that of the STMs' polytime ones.

▶ **Proposition 8.** For any poly-time STM $\mathcal{S}$ there is a polytime PTM $\mathcal{M}$ such that for every $\sigma\tau \in \mathbb{S}$:

$$\mu(\{\omega \in \mathbb{O} \mid N(\sigma\omega) = \tau\}) = \Pr[M(\sigma) = \tau]$$

and viceversa.

Proof. The claim can be restated as follows:

$$\forall\sigma, \tau.\mu(\{\eta \in \mathbb{B}^{\mathbb{N}} \mid N(\sigma, \eta) = \tau\}) = \mu(M(\sigma)^{-1}(\tau))$$

$$\forall\sigma, \tau.\mu(\{\eta \in \mathbb{B}^{\mathbb{N}} \mid N(\sigma, \eta) = \tau\}) = \mu(\{\eta \in \mathbb{B}^{\mathbb{N}} \mid Y_{M,\sigma}(\eta) = \tau\}).$$

Actually, we will show a stronger result: there is bijection I : STMs $\longrightarrow$ PTM such that:

$$\forall n \in \mathbb{N}.\{\eta \in \mathbb{B}^{\mathbb{N}} \mid \langle\sigma, q_0, \tau, \eta\rangle \triangleright^n_\delta \langle\tau, q, \psi, n\rangle\} = \{\eta \in \mathbb{B}^{\mathbb{N}} \mid X^{\langle\epsilon,q_0,\sigma\rangle}_{I(N),n}(\eta) = \langle\tau, q, \psi\rangle\} \tag{3}$$

which entails:

$$\{\eta \in \mathbb{B}^{\mathbb{N}} \mid N(\sigma, \eta) = \tau\} = \{\eta \in \mathbb{B}^{\mathbb{N}} \mid Y_{I(N),\sigma}(\eta) = \tau\}. \tag{4}$$

$$\begin{tikzcd}
c := \langle \sigma, q, \tau, y \rangle \arrow[r, "\vdash_\delta"] \arrow[d, "e_c"] & d := \langle \sigma, q, \tau, y \rangle \arrow[d, "e_c"] \\
s_c \in \mathbb{S} \arrow[r, "\forall \omega. step(s_c, e_t(\delta), \omega) = s_d"] & s_d \in \mathbb{S}
\end{tikzcd}$$

Figure 1 Behavior of the function *step*.

For this reason, it suffices to construct I and prove that (3) holds. I splits the function δ of N in such a way that transition is assigned to δ_0 if it matches the character $\mathbf{0}$ on the oracle-tape, otherwise it is assigned to δ_1. Observe that I is bijective, indeed, its inverse is a function as well, because it consists in a disjoint union. Claim (3) can be shown by induction on the number of steps required by N to compute its output value, the proof is standard, so we omit it.

◀

▶ **Proposition 9.** For every $f : \mathbb{S} \times \mathbb{B}^{\mathbb{N}} \longrightarrow Ss$ in **SFP**, there is a function $g : \mathbb{S} \times \mathbb{O} \longrightarrow \mathbb{S}$ in $\mathcal{POR}$ such that:

$$\forall x, y \in \mathbb{S}. \mu(\{\omega \in \mathbb{O} | g(x, \omega) = y\}) = \mu(\{\eta \in \mathbb{B}^{\mathbb{N}} | f(x, \eta) = y\}).$$

To prove the correspondence between the class of polytime STM computable function and $\mathcal{POR}$, we pass through the class of *finite-stream* TMs. These machines are defined analogously to STMs, but instead of an infinite stream of bits η, they employ a finite sequence of random bits as additional argument.

▶ **Lemma 22.** *For each $f \in$ **SFP** with time-bound $p \in$ POLY, there is a polytime* finite-stream *TM computable function h such that for any $\eta \in \mathbb{B}^{\mathbb{N}}$ and $x, y \in \mathbb{S}$,*

$$f(x, \eta) = h(x, \eta_{p(|x|)}).$$

Proof. Assume that $f \in$ **SFP**. For this reason there is a polytime STM, $\mathcal{S} = \langle \mathcal{Q}, q_0, \Sigma, \delta \rangle$, such that $f = f_{\mathcal{S}}$. Take the *Finite Stream Turing Machine* (FSTM) $\mathcal{S}'$ which is defined identically to $\mathcal{S}$. It holds that for any $k \in \mathbb{N}$ and some $\sigma, \tau, y' \in \mathbb{S}$,

$$\langle \boldsymbol{\epsilon}, q_0', x, y \rangle \triangleright^k_{\mathcal{S}} \langle \sigma, q, \tau, y' \rangle \quad \Leftrightarrow \quad \langle \boldsymbol{\epsilon}, q_0', x, y\eta \rangle \triangleright^k_{\mathcal{S}'} \langle \sigma, q, \tau, y'\eta \rangle.$$

Moreover, $\mathcal{S}'$ requires a number of steps which is exactly equal to the number of steps required by $\mathcal{S}$, and thus the complexity is preserved. We conclude the proof defining $h = f_{\mathcal{S}'}$. ◀

The next step is to show that each polytime FSTM computable function f corresponds to a function $g : \mathbb{S} \times \mathbb{S} \times \mathbb{O} \longrightarrow \mathbb{S}$ of $\mathcal{POR}$ which can be defined without recurring to Q.

▶ **Lemma 23.** *For any polytime FSTM computable function f and $x \in \mathbb{S}$, there is $g \in \mathcal{POR}$ such that $\forall x, y, \omega. f(x, y) = g(x, y, \omega)$. Moreover, if f is defined without recurring to Q, g can be defined without Q as well.*

A formal proof of Lemma 23 requires too much effort to be done extensively. In this work, we will simply mention the high-level structure of the proof. It relies on the following observations:

1. It is possible to encode FSTMs, together with configurations and their transition functions using strings, call these encodings $e_c \in \mathcal{POR}$ and e_t. Moreover, there is a function $\mathit{step} \in \mathcal{POR}$ which satisfies the simulation schema of Figure 1. The proof of this result is done by explicit definition of the functions e_c, e_t and *step*, proving the correctness of these entities with respect to the simulation schema above.
2. For each $f \in \mathcal{POR}$ and $x, y \in \mathbb{S}$, if there is a term $t(x)$ in $\mathcal{RL}$ which bounds the size of $f(x, \omega)$ for each possible input, then the function $m(z, x, \omega) = f^{|z|}(x, \omega)$ is in $\mathcal{POR}$ as well, moreover, if f is defined without recurring to Q, also m can be defined without recurring to Q. This is shown in Lemma 24.
3. Fixed a machine M, if $\sigma \in \mathbb{S}$ is a correct encoding of a configuration of M, for every ω, it holds that $|\mathit{step}(\sigma, \omega)| \leq |\sigma| + c$, for $c \in \mathbb{N}$ fixed once and forall.
4. If $c := e_c(\sigma, q, \tau, y, \omega)$ for some omega, then there is a function *dectape* such that $\forall \omega \in \mathbb{O}.\mathit{dectape}(x, \omega)$ is the longest suffix without occurrences of $\circledast$ of σ.

▶ **Lemma 24.** *For each $f : \mathbb{S}^{k+1} \times \mathbb{O} \longrightarrow \mathbb{S} \in \mathcal{POR}$, if there is a term $t \in \mathcal{RL}$ such that $\forall x, \vec{z}, \omega. f(x, \vec{z}, \omega)|_t = f(x, \vec{z}, \omega)$ then there is also a function $\mathit{sa}_{f,t} : \mathbb{S}^{k+2} \times \mathbb{O} \longrightarrow \mathbb{S}$ such that:*

$$\forall x, n \in \mathbb{S}, \omega \in \mathbb{O}. \mathit{sa}_{f,t}(x, n, \vec{z}, \omega) = \underbrace{f(f(f(x, \vec{z}, \omega), \vec{z}, \omega), \dots)}_{|n| \ \mathit{times}}.$$

Moreover, if f is defined without recurring to Q, $\mathit{sa}_{f,t}$ can be defined without Q as well.

Proof. Given $f \in \mathcal{POR}$ and $t \in \mathcal{L}_{\mathbb{PW}}$, let $\mathit{sa}_{f,t}$ be defined as follows:

$$\mathit{sa}_{f,t}(x, \boldsymbol{\epsilon}, \vec{z}, \omega) := x$$

$$\mathit{sa}_{f,t}(x, y\mathbf{b}, \vec{z}, \omega) := f\big(\mathit{sa}_{f,t}(x, y, \omega), \vec{z}, \omega\big)|_t$$

The correctness of *sa* comes as a direct consequence of its definition by induction on n. ◀

Combining these results, we are able to prove Lemma 23

Proof of Lemma 23 (Sketch). As a consequence of points (2) and (3), we obtain that: $k(x, n, y, \omega) = \mathit{step}^{|n|}(x, y, \omega)$ belongs to $\mathcal{POR}$ and can be defined without recurring to Q. As a consequence of (1) we have that:

$$k'(x, y, \omega) := k(e_c(x, q_0, \boldsymbol{\epsilon}, y, \omega), y, \omega)$$

belongs to $\mathcal{POR}$ as well and can be defined without recurring to Q. Finally, as a consequence of (4) and $\mathcal{POR}$'s closure under composition, there is a function g which returns the longest prefix of the leftmost projection of the output of k'. This function is exactly:

$$g(x, y, \omega) := \mathit{dectape}(k'(x, y, \omega), \omega).$$

◀

As another consequence of Lemma 23, we show the result we were aiming to: each function $f \in \mathbf{SFP}$ can be simulated by a function in $g \in \mathcal{POR}$, using as an additional input a polynomial prefix of f's oracle.

▶ **Corollary 10.** For each $f \in \mathbf{SFP}$ and polynomial time-bound $p \in \mathsf{POLY}$, there is a function $g \in \mathcal{POR}$ such that for any $\eta : \mathbb{N} \longrightarrow \mathbb{B}$, $\omega : \mathbb{N} \longrightarrow \mathbb{B}$ and $x \in \mathbb{S}$,

$$f(x, \eta) = g\big(x, \eta_{p(|x|)}, \omega\big).$$

Now, we need to establish that there is a function $e \in \mathcal{POR}$ which produces strings with the same distribution of the prefixes of the functions in $\mathbb{S}^{\mathbb{N}}$. Intuitively, this function is very simple: it extracts $|x|+1$ bits from ω and concatenates them in its output. The definition of the function e passes through a bijection $dyad : \mathbb{N} \longrightarrow \mathbb{S}$, called *dyadic representation* of a natural number. Thus, the function e can simply create the strings $\mathbf{1}^0, \mathbf{1}^1, \dots, \mathbf{1}^k$, and sample the function ω on the coordinates $dy(\mathbf{1}^0), dy(\mathbf{1}^1), \dots, dy(\mathbf{1}^k)$, concatenating the result in a string.

▶ **Definition 26.** The function $dyad : \mathbb{N} \longrightarrow \mathbb{S}$ associates each $n \in \mathbb{N}$ to the string obtained stripping the left-most bit from the binary representation of $n+1$.

▶ Remark 25. There is a $\mathcal{POR}$ function $dy : \mathbb{S} \times \mathbb{O} \longrightarrow \mathbb{S}$ such that $\forall \sigma \in \mathbb{S}.\forall \omega \in \mathbb{O}.dy(\sigma, \omega) = dyad(\mathbf{1}^{|\sigma|})$.

The construction of this function is not much interesting to the aim of our proof, so we omit it.

▶ **Definition 27.** Let $e : \mathbb{S} \times \mathbb{O} \longrightarrow \mathbb{S}$ be defined as follows:

$$e(\boldsymbol{\epsilon}, \omega) = \boldsymbol{\epsilon};$$
$$e(x\mathbf{b}, \omega) = e(x, \omega)Q\big(dy(x, \omega), \omega\big)|_{x\mathbf{b}}.$$

▶ **Lemma 26** (Correctness of e)**.** *For any $\sigma \in \mathbb{S}$ and $i \in \mathbb{N}$, if $|\sigma| = i+1$, for any $j \leq i \in \mathbb{N}$ and $\omega \in \mathbb{O}$, (i) $e(\sigma, \omega)(j) = \omega(dy(\mathbf{1}^j, \omega))$ and (ii) the length of $e(\sigma, \omega)$ is exactly $i+1$.*

Proof. Both claims are proved by induction on σ. The latter is trivial, whereas the former requires some more effort:

- $\boldsymbol{\epsilon}$ The claim comes from vacuity of the premise $|\sigma| = i+1$.
- $\tau\mathbf{b}$ It holds that: $e(\tau\mathbf{b}, \omega)(j) = e(\tau, \omega)Q\big(dy(\tau, \omega), \omega\big) = e(\tau, \omega)\omega\big(dy(\tau, \omega)\big)$. By (ii), for $j = i+1$, the j-th element of $e(\tau\mathbf{b}, \omega)$ is exactly $Q\big(dy(\tau, \omega), \omega\big)$, which is equal to $\omega(dy(\tau, \omega))$, in turn equal to $\omega(dy(\mathbf{1}^j, \omega))$ (by Remark 25). For smaller values of j, the first claim is a consequence of the definition of e together with IH.

◀

▶ **Definition 28.** We define $\sim_{dy}$ as the smallest relation in $\mathbb{O} \times \mathbb{B}^{\mathbb{N}}$ such that:

$$\eta \sim_{dy} \omega \leftrightarrow \forall n \in \mathbb{N}.\eta(n) = \omega(dy(\mathbf{1}^{n+1}, \omega)).$$

▶ **Lemma 27.** *It holds that:*

- *$\forall \eta \in \mathbb{B}^{\mathbb{N}}.\exists! \omega \in \mathbb{O}.\eta \sim_{dy} \omega$;*
- *$\forall \omega \in \mathbb{O}.\exists! \eta \in \mathbb{B}^{\mathbb{N}}.\eta \sim_{dy} \omega$.*

Proof. The proofs of the two claims are very similar. Since $dyad$ is a bijection, applying Remark 25, we obtain the existence of an ω which is in relation with η. Now suppose that there are ω_1, ω_2 both in relation with η but they are different. Then, there is a $\sigma \in \mathbb{S}$, such that $\omega_1(\sigma) \neq \omega_2(\sigma)$ and, by Remark 25, $dy(\sigma, \omega_1) = dy(\sigma, \omega_2)$ which entails that $\eta(k) \neq \eta(k)$ for some k, that is a contradiction. ◀

▶ **Corollary 11.** The relation $\sim_{dy}$ is a bijection.

Proof. Consequence of Lemma 27. ◀

▶ **Lemma 28.**

$$\eta \sim_{dy} \omega \rightarrow \forall n \in \mathbb{N}.\eta_n = e(1^{n+1}, \omega).$$

Proof. By contraposition: suppose $\eta_n \neq e(\underline{n}_{\mathbb{N}}, \omega)$. As a consequence of the correctness of e (Lemma 26), there is an $i \in \mathbb{N}$ such that $\eta(i) \neq \omega(dy(\underline{i}_{\mathbb{N}}, \omega))$, which is a contradiction. ◀

We can finally conclude the proof of Proposition 9.

Proof of Proposition 9. From Corollary 10, we know that there is a function $f' \in \mathcal{POR}$, and a $p \in \mathsf{POLY}$ such that:

$$\forall x, y \in \mathbb{S}.\forall \eta.\forall \omega. y = \eta_{p(x)} \to f(x, \eta) = f'(x, y, \omega). \tag{$*$}$$

Fixed an $\overline{\eta} \in \{\eta \in \mathbb{B}^{\mathbb{N}} \mid f(x, \eta) = y\}$, its image with respect to $\sim_{dy}$ is in $\{\omega \in \mathbb{O} \mid f'(x, e(p'(s(x, \omega), \omega), \omega), \omega) = y\}$, where s is the $\mathcal{POR}$-function computing $\mathbf{1}^{|x|}$. Indeed, by Lemma 28, it holds that $\overline{\eta}_{p(x)} = e(p(\mathit{size}(x, \omega), \omega)$, where p' is the $\mathcal{POR}$-function computing the polynomial p, defined without recurring to Q. By ($*$), we prove the claim. It also holds that, given a fixed $\overline{\omega} \in \{\omega \in \mathbb{O} \mid f'(x, e(p'(\mathit{size}(x, \omega), \omega), \omega), \omega) = y\}$, its pre-image with respect to $\sim_{dy}$ is in $\{\eta \in \mathbb{B}^{\mathbb{N}} \mid f(x, \eta) = y\}$. The proof is analogous to the one we showed above. Now, since $\sim_{dy}$ is a bijection between the two sets: $\mu\big(\{\eta \in \mathbb{B}^{\mathbb{N}} \mid f(x, \eta) = y\}\big) = \mu\big(\{\omega \in \mathbb{O} \mid f'(x, e(p(\mathit{size}(x, \omega), \omega), \omega), \omega) = y\}\big)$, which concludes the proof. ◀

B.2 From $\mathcal{POR}$ to SFP

We start defining the imperative language $\mathbf{SIFP_{RA}}$ and proving its polytime programs equivalent to $\mathcal{POR}$. To do so, we first introduce the definition of $\mathbf{SIFP_{RA}}$ and its *big-step* semantics.

▶ **Definition 29** (Correct programs of $\mathbf{SIFP_{RA}}$)**.** The language of $\mathbf{SIFP_{RA}}$ programs is $\mathcal{L}(\mathsf{Stm}_{\mathbf{RA}})$, i.e. the set of strings produced by the non-terminal symbol $\mathsf{Stm}_{\mathbf{RA}}$ defined by:

$$\begin{aligned}
\mathsf{Id} &::= X_i \mid Y_i \mid S_i \mid R \mid Q \mid Z \mid T \qquad i \in \mathbb{N}\\
\mathsf{Exp} &::= \epsilon \mid \mathsf{Exp.0} \mid \mathsf{Exp.1} \mid \mathsf{Id} \mid \mathsf{Exp} \sqsubseteq \mathsf{Id} \mid \mathsf{Exp} \wedge \mathsf{Id} \mid \neg\mathsf{Exp}\\
\mathsf{Stm}_{\mathbf{RA}} &::= \mathsf{Id} \leftarrow \mathsf{Exp} \mid \mathsf{Stm}_{\mathbf{RA}}; \mathsf{Stm}_{\mathbf{RA}} \mid \mathtt{while}(\mathsf{Exp})\{\mathsf{Stm}\}_{\mathbf{RA}} \mid \mathtt{Flip}(\mathsf{Exp})
\end{aligned}$$

The *big-step* semantics associated to the language of the $\mathbf{SIFP_{RA}}$ programs relies on the notion of *Store*, which for us is a function $\Sigma : \mathsf{Id} \rightharpoonup \{\mathsf{0}, \mathsf{1}\}^*$.

We define the updating of a store Σ with a mapping from $y \in \mathsf{Id}$ to $\tau \in \{\mathsf{0}, \mathsf{1}\}^*$ as:

$$\Sigma[y \leftarrow \tau](x) := \begin{cases} \tau & \text{if } x = y \\ \Sigma(x) & \text{otherwise.} \end{cases}$$

▶ **Definition 30** (Semantics of **SIFP** expressions)**.** The semantics of an expression $E \in \mathcal{L}(\mathsf{Exp})$ is the smallest relation $\rightharpoonup: \mathcal{L}(\mathsf{Exp}) \times (\mathsf{Id} \longrightarrow \{\mathsf{0}, \mathsf{1}\}^*) \times \mathbb{O} \times \{\mathsf{0}, \mathsf{1}\}^*$ closed under the following rules:

$$\frac{}{\langle \epsilon, \Sigma \rangle \rightharpoonup \epsilon} \qquad \frac{\langle e, \Sigma \rangle \rightharpoonup \sigma}{\langle e.\mathsf{0}, \Sigma \rangle \rightharpoonup \sigma \frown \mathsf{0}} \qquad \frac{\langle e, \Sigma \rangle \rightharpoonup \sigma}{\langle e.\mathsf{1}, \Sigma \rangle \rightharpoonup \sigma \frown \mathsf{1}}$$

$$\frac{\langle e, \Sigma \rangle \rightharpoonup \sigma \qquad \Sigma(\mathsf{Id}) = \tau \qquad \sigma \subseteq \tau}{\langle e \sqsubseteq \mathsf{Id}, \Sigma \rangle \rightharpoonup \mathsf{1}} \qquad \frac{\langle e, \Sigma \rangle \rightharpoonup \sigma \qquad \Sigma(\mathsf{Id}) = \tau \qquad \sigma \not\subseteq \tau}{\langle e \sqsubseteq \mathsf{Id}, \Sigma \rangle \rightharpoonup \mathsf{0}}$$

$$\frac{\Sigma(\mathsf{Id}) = \sigma}{\langle \mathsf{Id}, \Sigma \rangle \rightharpoonup \sigma} \qquad \frac{\mathsf{Id} \notin \mathit{dom}(\Sigma)}{\langle \mathsf{Id}, \Sigma \rangle \rightharpoonup \epsilon} \qquad \frac{\langle e, \Sigma \rangle \rightharpoonup \mathsf{0}}{\langle \neg e, \Sigma \rangle \rightharpoonup \mathsf{1}} \qquad \frac{\langle e, \Sigma \rangle \rightharpoonup \sigma \qquad \sigma \neq \mathsf{0}}{\langle \neg e, \Sigma \rangle \rightharpoonup \mathsf{0}}$$

$$\frac{\langle e, \Sigma\rangle \rightharpoonup \mathsf{1} \qquad \Sigma(\mathsf{Id}) = \mathsf{1}}{\langle e \wedge \mathsf{Id}, \Sigma\rangle \rightharpoonup \mathsf{1}} \qquad \frac{\langle e, \Sigma\rangle \rightharpoonup \sigma \qquad \Sigma(\mathsf{Id}) = \tau \qquad \sigma \neq \mathsf{1} \wedge \tau \neq \mathsf{1}}{\langle e \wedge \mathsf{Id}, \Sigma\rangle \rightharpoonup \mathsf{0}}$$

▶ **Definition 31** (big-step Operational Semantics of $\mathbf{SIFP_{RA}}$)**.** The semantics of a program $P \in \mathcal{L}(\mathsf{Stm}_{\mathbf{RA}})$ is the smallest relation $\triangleright \subseteq \mathcal{L}(\mathsf{Stm}_{\mathbf{RA}}) \times (\mathsf{Id} \longrightarrow \{\mathsf{0}, \mathsf{1}\}^*) \times \mathbb{O} \times (\mathsf{Id} \longrightarrow \{\mathsf{0}, \mathsf{1}\}^*)$ closed under the following rules:

$$\frac{\langle e, \Sigma\rangle \rightharpoonup \sigma}{\langle \mathsf{Id} \leftarrow e, \Sigma, \omega\rangle \triangleright \Sigma[\mathsf{Id} \leftarrow \sigma]} \qquad \frac{\langle s, \Sigma, \omega\rangle \triangleright \Sigma' \qquad \langle t, \Sigma', \omega\rangle \triangleright \Sigma''}{\langle s; t, \Sigma, \omega\rangle \triangleright \Sigma''}$$

$$\frac{\langle e, \Sigma\rangle \rightharpoonup \mathsf{1} \qquad \langle s, \Sigma, \omega\rangle \triangleright \Sigma' \qquad \langle \mathtt{while}(e)\{s\}, \Sigma', \omega\rangle \triangleright \Sigma''}{\langle \mathtt{while}(e)\{s\}, \Sigma, \omega\rangle \triangleright \Sigma''} \qquad \frac{\langle e, \Sigma\rangle \rightharpoonup \sigma \qquad \sigma \neq \mathsf{1}}{\langle \mathtt{while}(e)\{s\}, \Sigma, \omega\rangle \triangleright \Sigma}$$

$$\frac{\langle e, \Sigma\rangle \rightharpoonup \sigma \qquad \omega(\sigma) = b}{\langle \mathtt{Flip}(e), \Sigma, \omega\rangle \triangleright \Sigma[R \leftarrow b]}$$

This semantics allows us to associate each $\mathbf{SIFP_{RA}}$ program to the function it evaluates:

▶ **Definition 32** (Function evaluated by a $\mathbf{SIFP_{RA}}$ program)**.** We say that the function evaluated by a correct $\mathbf{SIFP_{RA}}$ program P is $[\![\cdot]\!] : \mathcal{L}(\mathsf{Stm}_{\mathbf{RA}}) \longrightarrow (\mathbb{S}^n \times \mathbb{O} \longrightarrow \mathbb{S})$, defined as below[6]:

$$[\![P]\!] := \lambda x_1, \dots, x_n, \omega. \triangleright (\langle P, [][X_1 \leftarrow x_1], \dots, [X_n \leftarrow x_n], \omega\rangle)(R).$$

Observe that, among all the different registers, the register R is meant to contain the value computed by the program at the end of its execution, similarly the $\{X_i\}_{i\in\mathbb{N}}$ registers are used to store the function's inputs. The correspondence between $\mathcal{POR}$ and $\mathbf{SIFP_{RA}}$ can be stated as follows:

▶ **Lemma 29** (Implementation of $\mathcal{POR}$ in $\mathbf{SIFP_{RA}}$)**.** *For every function $f \in \mathcal{POR}$, there is a polytime $\mathbf{SIFP_{RA}}$ program P such that: $\forall x_1, \dots x_n. [\![P]\!](x_1, \dots, x_n, \omega) = f(x_1, \dots, x_n, \omega)$. Moreover, if f can be defined without recurring to Q, then P does not contain any $\mathtt{Flip}(e)$ statement.*

The proof of this result is quite simple: it relies on the fact that it is possible to associate to each $\mathcal{POR}$ function an equivalent polytime program, and on the observation that it is possible to compose them and to implement bounded recursion on notation in $\mathbf{SIFP_{RA}}$ with a polytime overhead.

Proof Sketch of Lemma 29. For each function $f \in \mathcal{POR}$ we define a program $\mathfrak{L}_f$ such that $[\![\mathfrak{L}_f]\!](x_1, \dots, x_n) = f(x_1, \dots, x_n)$ The correctness of $\mathfrak{L}_f$ is given by the following invariant properties:

- The result of the computation is stored in R.
- The inputs are stored in the registers of the group X.
- The function $\mathfrak{L}.$ does not change the values it accesses as input.

We define the function $\mathfrak{L}_f$ as follows: $\mathfrak{L}_E := R \leftarrow \epsilon$; $\mathfrak{L}_{S_0} := R \leftarrow X_0.\mathsf{0}$; $\mathfrak{L}_{S_1} := R \leftarrow X_0.\mathsf{1}$; $\mathfrak{L}_{P^n_i} := R \leftarrow X_i$; $\mathfrak{L}_C := R \leftarrow X_1 \sqsubseteq X_2$; $\mathfrak{L}_Q := \mathtt{Flip}(X_1)$. The correctness of these base cases is trivial. Moreover, it is simple to see that the only translation containing $\mathtt{Flip}(e)$ for some

[6] Instead of the infix notation for $\triangleright$, we will use its prefixed notation. So, the notation express the store associated to the P, Σ and ω by $\triangleright$. Moreover, notice that we employed the same function symbol $\triangleright$ to denote two distinct functions: the *big-step* operational semantics of $\mathbf{SIFP_{RA}}$ programs and the *big-step* operational semantics of $\mathbf{SIFP_{LA}}$ programs

$e \in \mathcal{L}(\mathsf{Exp})$ is the translation of Q. The encoding of the composition and of the bounded recursion are a little more convoluted: the proof of their correctness requires a conspicuous amount of low-level definitions and technical results, whose presentation is not the aim of this work. ◀

The next step is to sow that every $\mathbf{SIFP_{RA}}$ program is equivalent to a $\mathbf{SIFP_{LA}}$ program in the sense of Lemma 30.

▶ **Lemma 30.** *For each total program $P \in \mathbf{SIFP_{RA}}$ there is a $Q \in \mathbf{SIFP_{LA}}$ such that:*

$$\forall x, y.\mu\left(\{\omega \in \mathbb{B}^{\mathbb{S}} | [\![P]\!](x, \omega) = y\}\right) = \mu\left(\{\eta \in \mathbb{B}^{\mathbb{N}} | [\![Q]\!](x, \eta) = y\}\right).$$

Moreover, if P is polytime Q is polytime, too.

Before delving into the details of the proof of this Lemma, we must define the language $\mathbf{SIFP_{LA}}$ together with its standard semantics:

▶ **Definition 33** ($\mathbf{SIFP_{LA}}$)**.** The language of the $\mathbf{SIFP_{LA}}$ programs is $\mathcal{L}(\mathsf{Stm}_{\mathbf{LA}})$, i.e. the set of strings produced by the non-terminal symbol $\mathsf{Stm}_{\mathbf{LA}}$ described as follows:

$$\mathsf{Stm}_{\mathbf{LA}} ::= \mathsf{Id} \leftarrow \mathsf{Exp} \mid \mathsf{Stm}_{\mathbf{LA}}; \mathsf{Stm}_{\mathbf{LA}} \mid \mathtt{while}(\mathsf{Exp})\{\mathsf{Stm}\}_{\mathbf{LA}} \mid \mathtt{RandBit}()$$

▶ **Definition 34** (Big Step Operational Semantics of $\mathbf{SIFP_{LA}}$)**.** The semantics of a program $P \in \mathcal{L}(\mathsf{Stm}_{\mathbf{LA}})$ is the smallest relation $\triangleright \subseteq \left(\mathcal{L}(\mathsf{Stm}_{\mathbf{LA}}) \times (\mathsf{Id} \longrightarrow \{\mathtt{0},\mathtt{1}\}^*) \times \mathbb{B}^{\mathbb{N}}\right) \times \left((\mathsf{Id} \longrightarrow \{\mathtt{0},\mathtt{1}\}^*) \times \mathbb{B}^{\mathbb{N}}\right)$ closed under the following rules:

$$\frac{\langle e, \Sigma\rangle \rightharpoonup \sigma}{\langle \mathsf{Id} \leftarrow e, \Sigma, \eta\rangle \triangleright \langle \Sigma[\mathsf{Id} \leftarrow \sigma], \eta\rangle} \qquad \frac{\langle s, \Sigma, \eta\rangle \triangleright \langle \Sigma', \eta'\rangle \qquad \langle t, \Sigma', \eta\rangle \triangleright \langle \Sigma'', \eta''\rangle}{\langle s; t, \Sigma, \eta\rangle \triangleright \langle \Sigma'', \eta''\rangle}$$

$$\frac{\langle e, \Sigma\rangle \rightharpoonup \mathtt{1} \qquad \langle s, \Sigma, \eta\rangle \triangleright \langle \Sigma', \eta'\rangle \qquad \langle \mathtt{while}(e)\{s\}, \Sigma', \eta\rangle \triangleright \langle \Sigma'', \eta''\rangle}{\langle \mathtt{while}(e)\{s\}, \Sigma, \eta\rangle \triangleright \langle \Sigma'', \eta''\rangle}$$

$$\frac{\langle e, \Sigma\rangle \rightharpoonup \sigma \qquad \sigma \neq \mathtt{1}}{\langle \mathtt{while}(e)\{s\}, \Sigma, \eta\rangle \triangleright \langle \Sigma, \eta\rangle} \qquad \frac{}{\langle \mathtt{RandBit}(), \Sigma, \mathtt{b}\eta\rangle \triangleright \langle \Sigma[R \leftarrow \mathtt{b}], \eta\rangle}$$

In particular, we prove Lemma 30 showing that $\mathbf{SIFP_{RA}}$ can be simulated in $\mathbf{SIFP_{LA}}$ with respect to two novel *small-step* semantic relations ($\rightsquigarrow_{\mathbf{LA}}, \rightsquigarrow_{\mathbf{RA}}$) derived splitting the *big-step* semantics into small transitions, one per each $\cdot;\cdot$ instruction. Intuitively, the idea behind this novel semantics is to enrich the *big-step* operational semantic with some pieces of information necessary to build a proof that it is possible to each $\mathbf{SIFP_{LA}}$ instruction by meas of a sequence of $\mathbf{SIFP_{RA}}$ instructions preserving the distribution of the values in the register R, i.e. that storing the result. In particular we enrich the configurations of $\mathbf{SIFP_{LA}}$'s and $\mathbf{SIFP_{RA}}$'s *small-step* operational semantics with a list Ψ containing pairs (x, b). In the case of $\mathbf{SIFP_{LA}}$, this list is meant to keep track of the calls to the primitive $\mathtt{Flip}(x)$ and their result b. While, on the side of $\mathbf{SIFP_{RA}}$, the x-th call of the primitive $\mathtt{RandBit}()$ causes the pair (x, b) to be added on top of the list.

This is done to keep track of the accesses to the random tapes done by the simulated and the simulator programs. On the side of $\mathbf{SIFP_{RA}}$ it is possible to show that this table is stored in a specific register. This register plays an important role in the simulation of the $\mathtt{Flip}(e)$ instructions. In particular, this is done as follows:

- At each simulated query $\mathtt{Flip}(e)$, the destination program looks up the associative table;
- If it finds the queried coordinate e within a pair (e, b), it returns b, otherwise:
  - It reduces $\mathtt{Flip}(e)$ to a call of $\mathtt{RandBit}()$ which outputs either $\mathtt{b} = \mathbf{0}$ or $\mathtt{b} = \mathbf{1}$.

- It records the pair $\langle e, \mathtt{b}\rangle$ in the associative table and stores $\mathtt{b}$ in R.

Even in this case the construction of the proof is convoluted, but we believe that it is not too much of a problem to see, at least intuitively, that this kind of simulation preserves the distributions of strings computed by the original program. This concludes Lemma 30.
The next step is to show that $\mathbf{SIFP_{LA}}$ can be reduced to $\mathbf{SFP_{OD}}$: the class corresponding to $\mathbf{SFP}$ defined on a variation of the Stream Machines which are capable to read characters from the oracle tape *on-demand*, i.e. only on those states $q \in \mathcal{Q}_\natural \subseteq \mathcal{Q}$. The transition function of the $\mathbf{SFP_{OD}}$ is a function $\delta : \hat{\mathbb{B}} \times \mathcal{Q} \times \left(\hat{\mathbb{B}} \cup \{\natural\}\right) \times \mathbb{B} \longrightarrow \hat{\mathbb{B}} \times \mathcal{Q} \times \hat{\mathbb{B}} \times \{L, R\}$ which for all the states in $\mathcal{Q}_\natural$ is labeled with the symbol $\natural$ instead of a Boolean value. This peculiarity will be employed in Definition 35 to distinguish those configurations causing the oracle tape to shift to the others.

We do not show the reduction from $\mathbf{SIFP_{LA}}$ to $\mathbf{SFP_{OD}}$ extensively because this kind of reductions are cumbersome and, in literature, it is common to avoid their formal definition on behalf of a more readable presentation. For this reason, we only describe informally but exhaustively how to build the *on-demand* stream machine which corresponds to a generic program $P \in \mathcal{L}(\mathsf{Stm}_{\mathbf{LA}})$. The correspondence between $\mathbf{SFP_{OD}}$ is expressed by the following Proposition:

▶ **Proposition 10.** For every $P \in \mathcal{L}(\mathsf{Stm}_{\mathbf{LA}})$ there is a $M_P \in \mathbf{SFP}$ such that for every $x \in \mathbb{S}$ and $\eta \in \mathbb{B}^{\mathbb{S}}$, $P(x, \eta) = P(x, \eta)$. Moreover, if P is polytime, then M_P is polytime.

Proof. The construction relies on the fact that it is possible to implement a $\mathbf{SIFP_{LA}}$ program by means of a multi-tape *on-demand stream machine* which uses a tape to store the values of each register, plus an additional tape containing the partial results obtained during the evaluation of the expressions and another tape containing η. We denote the tape used for storing the result coming from the evaluation of the expressions with e.

The machine works thanks to some invariant properties:

- On each tape, the values are stored to the immediate right of the head.
- The result of the last expression evaluated is stored on the e tape to the immediate right of the head.

The value of a $\mathbf{SIFP}$ expression can be easily computed using the e tape. We show it by induction on the syntax of the expression:

- Each access to the value stored in a register basically consist in a copy of the content of the corresponding tape to the e tape, which is a simple operation, due to the invariant properties properties mentioned above.
- Concatenations ($f.\mathtt{0}$ and $f.\mathtt{1}$) are easily implemented by the addition of a character at the end of the e tape which contains the value of f, as stated by the induction hypothesis on the invariant properties.
- The binary expressions are non-trivial, but since one of the two operands is a register identifier, the machine can directly compare e with the tape which corresponding to the identifier, and to replace the content of e with the result of the comparison, which in all cases $\mathbf{0}$ or $\mathbf{1}$.

All these operations can be implemented without consuming any character on the oracle tape and with linear time with respect to the size of the expression's value. To each statement s_i, we assign a sequence of machine states, $q^I_{s_i}, q^1_{s_i}, q^2_{s_i}, \dots, q^F_{s_i}$.

- Assignments consist in a copy of the value in e to the tape corresponding to the destination register and a deletion of the value on e by replacing its symbols with $\circledast$ characters. This can be implemented without consuming any character on the oracle tape.

- The sequencing operation $s;t$ can be implemented inserting in δ a composed transition from q_s^F to q_t^I, which does not consume the oracle tape.
- A $\texttt{while}()\{\text{s}\}$tatement $s := \texttt{while}(f)\{t\}$ requires the evaluation of f and then passing to the evaluation of t, if $f \rightharpoonup \texttt{1}$, or stepping to the next transition if it exists and $f \not\rightharpoonup \texttt{1}$. After the evaluation of the body, the machine returns to the initial state of this statement, namely: q_s^I.
- A $\texttt{RandBit}()$ statement is implemented consuming a character on the tape and copying its value on the tape which corresponds to the register R.

The following invariant properties hold at the beginning of the execution and are kept true throughout M_P's execution. In particular, if we assume P to be polytime, after the simulation of each statement, it holds that:

- The length of the non blank portion of the first tapes corresponding to the register is polynomially bounded because their contents are precisely the contents of P's registers, which are polynomially bounded as a consequence of the hypotheses on their polynomial time complexity.
- The head of all the tapes corresponding to the registers point to the leftmost symbol of the string thereby contained.

It is well-known that the reduction of the number of tapes on a polytime Turing Machine comes with a polynomial overhead in time; for this reason, we can conclude that the polytime *multi-tape* on-demand stream machine we introduced above can be *shrinked* to a polytime *canonical* on-demand stream machine. This concludes the proof. ◀

It remains to show that each on-demand stream machine can be reduced to an equivalent STM.

▶ **Lemma 31.** *For every $\mathcal{S} = \langle \mathcal{Q}, \mathcal{Q}_\natural, \Sigma, \delta, q_0\rangle \in \mathbf{SFP_{OD}}$, the machine $\mathcal{S}' = \langle \mathcal{Q}, \Sigma, H(\delta), q_0\rangle \in$ **SFP** is such that for every $n \in \mathbb{N}$, for every configuration of $\mathcal{S}$ $\langle \sigma, q, \tau, \eta\rangle$ and for every $\sigma', \tau' \in \mathbb{S}, q \in \mathcal{Q}$:*

$$\mu\left(\{\eta \in \mathbb{B}^{\mathbb{N}} | \exists \eta'.\langle \sigma, q, \tau, \eta\rangle \triangleright_\delta^n \langle \sigma', q', \tau', \eta'\rangle\}\right) = \mu\left(\{\chi \in \mathbb{B}^{\mathbb{N}} | \exists \chi'.\langle \sigma, q, \tau, \xi\rangle \triangleright_{H(\delta)}^n \langle \sigma', q', \tau', \chi'\rangle\}\right).$$

Even in this case, the proof relies on a reduction. In particular, we show that given an on-demand stream machine $\mathcal{S}$ it is possible to build a stream machine $\mathcal{S}'$ which is equivalent to $\mathcal{S}$. Intuitively, the encoding from an *on-demand* stream machine $\mathcal{S}$ to an ordinary stream machine takes the transition function δ of $\mathcal{S}$ and substitutes each transition not causing the oracle tape to shift — i.e. tagged with $\natural$ — with two distinct transitions, so to match both $\texttt{0}$ and $\texttt{1}$ on the tape storing ω. This causes the resulting machine to reach the same target state with the same behavior n the work tape, and to shift to the right the head on the oracle tape.

▶ **Definition 35** (Encoding from On-Demand to Canonical Stream Machines)**.** We define the encoding from an On-Demand Stream Machine to a Canonical Stream Machine as below:

$$H := \langle \mathbb{Q}, \Sigma, \delta, q_0\rangle \mapsto \left\langle \mathbb{Q}, \Sigma, \bigcup \Delta_H(\delta), q_0\right\rangle.$$

where Δ_H is defined as follows:

$$\Delta_H(\langle p, c_r, \mathbf{0}, q, c_w, d\rangle) := \{\langle p, c_r, \mathbf{0}, q, c_w, d\rangle\}$$
$$\Delta_H(\langle p, c_r, \mathbf{1}, q, c_w, d\rangle) := \{\langle p, c_r, \mathbf{1}, q, c_w, d\rangle\}$$
$$\Delta_H(\langle p, c_r, \natural, q, c_w, d\rangle) := \{\langle p, c_r, \mathbf{0}, q, c_w, d\rangle, \langle p, c_r, \mathbf{1}, q, c_w, d\rangle\}.$$

Proof of Lemma 31. The definition of $\triangleright^n_\delta$ allows us to rewrite the statement $\exists \eta'.\langle \sigma, q, \tau, \eta\rangle \triangleright^n_\delta \langle \sigma', q', \tau', \eta'\rangle$ as:

$$\begin{aligned}
&\exists \eta', \eta'' \in \mathbb{B}^{\mathbb{N}}.\exists c_1, \dots, c_k.\\
&\langle \sigma, q, \tau, c_1c_2\dots c_k\eta'\rangle \triangleright^{n_1}_\delta \langle \sigma_1, q_{i_1}, \tau_1, c_1c_2\dots c_k\eta'\rangle \triangleright^1_\delta \langle \sigma'_1, q'_{i_1}, \tau_1, c_2\dots c_k\eta'\rangle \wedge\\
&\langle \sigma'_1, q'_{i_1}, \tau_1, c_2\dots c_k\eta'\rangle \triangleright^{n_2}_\delta \langle \sigma_2, q_{i_2}, \tau_2, c_2\dots c_k\eta'\rangle \triangleright^1_\delta \langle \sigma'_2, q'_{i_2}, \tau'_2, c_3\dots c_k\eta'\rangle \wedge\\
&\langle \sigma'_2, q'_{i_2}, \tau'_2, c_3\dots c_k\eta'\rangle \triangleright^{n_3}_\delta \dots \triangleright^{n_{k+1}}_\delta \langle \sigma', q', \tau', \eta'\rangle.
\end{aligned}$$

The claim can be rewritten as follows:

$$\begin{aligned}
&\exists \eta'', c_1, \dots, c_k \in \mathbb{B}.\exists n_1, \dots, n_{k+1} \in \mathbb{N}.\forall \xi_1, \dots, \xi_{k+1} \in \mathbb{S}.|\xi_1| = n_1 \wedge \dots |\xi_{k+1}| = n_{k+1} \wedge\\
&\langle \sigma, q, \tau, c_1c_2\dots c_k\eta'\rangle \triangleright^{n_1}_\delta \langle \sigma_1, q_{i_1}, \tau_1, c_1c_2\dots c_k\eta'\rangle \triangleright^1_\delta \langle \sigma'_1, q'_{i_1}, \tau_1, c_2\dots c_k\eta'\rangle \wedge\\
&\langle \sigma'_1, q'_{i_1}, \tau_1, c_2\dots c_k\eta'\rangle \triangleright^{n_2}_\delta \langle \sigma_2, q_{i_2}, \tau_2, c_2\dots c_k\eta'\rangle \triangleright^1_\delta \langle \sigma'_2, q'_{i_2}, \tau'_2, c_3\dots c_k\eta'\rangle \wedge\\
&\langle \sigma'_2, q'_{i_2}, \tau'_2, c_3\dots c_k\eta'\rangle \triangleright^{n_3}_\delta \dots \triangleright^{n_{k+1}}_\delta \langle \sigma', q', \tau', \eta'\rangle\\
&\iff\\
&\langle \sigma, q, \tau, \xi_1c_1\xi_2c_2\dots c_k\xi_{k+1}\eta''\rangle \triangleright^{n_1}_{H(\delta)} \langle \sigma_1, q_{i_1}, \tau_1, c_1\xi_2c_2\dots c_k\xi_{k+1}\eta''\rangle \triangleright^1_{H(\delta)} \langle \sigma'_1, q'_{i_1}, \tau_1, \xi_2c_2\dots c_k\xi_{k+1}\eta''\rangle \wedge\\
&\langle \sigma'_1, q'_{i_1}, \tau_1, \xi_2c_2\dots c_k\xi_{k+1}\eta''\rangle \triangleright^{n_2}_{H(\delta)} \langle \sigma_2, q_{i_2}, \tau_2, c_2\dots c_k\xi_{k+1}\eta''\rangle \triangleright^1_{H(\delta)} \langle \sigma'_2, q'_{i_2}, \tau'_2, \xi_3c_3\dots c_k\xi_{k+1}\eta''\rangle \wedge\\
&\langle \sigma'_2, q'_{i_2}, \tau'_2, \xi_3c_3\dots c_k\eta''\rangle \triangleright^{n_3}_{H(\delta)} \dots \triangleright^{n_{k+1}}_{H(\delta)} \langle \sigma', q', \tau', \eta''\rangle.
\end{aligned}$$

Intuitively, this holds because it suffices to take the n_is as the length of longest sequence of non-shifting transitions of the on-demand stream machine and the correspondence can be proven by induction on the number of steps of each formula in the conjunction. Thus, we can express the sets of the claim as follows:

$$\{\eta \in \mathbb{B}^{\mathbb{N}} | \exists \eta'.\langle \sigma, q, \tau, \eta\rangle \triangleright^n_\delta \langle \sigma', q', \tau', \eta'\rangle\} = \{\eta \in \mathbb{B}^{\mathbb{N}} | \forall 0 \le i < k.\eta(i) = c_i\rangle\}$$
$$\{\chi \in \mathbb{B}^{\mathbb{N}} | \exists \chi'.\langle \sigma, q, \tau, \xi\rangle \triangleright^n_{H(\delta)} \langle \sigma', q', \tau', \chi'\rangle\} = \{\chi \in \mathbb{B}^{\mathbb{N}} | \forall 1 \le i \le k.\chi(n_i + i) = c_i \wedge \chi(0) = c_1\rangle\}.$$

The conclusion comes because both these sets are cylinders with the same measure. ◀

▶ **Proposition 32** (From $\mathcal{POR}$ to **SFP**). *For any $f : \mathbb{S}^k \times \mathbb{B}^{\mathbb{S}} \to \mathbb{S}$ in $\mathcal{POR}$ there exists a function $f^\sharp : \mathbb{S}^k \times \mathbb{B}^{\mathbb{N}} \to \mathbb{S}$ such that for all $n_1, \dots, n_k, m \in \mathbb{S}$,*

$$\mu\big(\{\eta \in \mathbb{B}^{\mathbb{N}} \mid f(n_1, \dots, n_k, \eta) = m\}\big) = \mu\big(\{\omega \in \mathbb{O} \mid f^\sharp(n_1, \dots, n_k, \omega) = m\}\big).$$

Proof. This result is a consequence of Lemma 29, Lemma 30, Proposition 10 and Lemma 31. ◀

C Proofs from Section 5

C.1 The Randomized Algorithm

In order to show that the formula G characterizes a randomized algorithm with *low-entropy*, namely, point (2) of Definition 9, we need more details on the internal structure of G. To do

$$\begin{tikzcd}
G_1 \arrow[r, dashed] \arrow[d, "\text{Realizability}"] & G_v \\
g_1 \arrow[r, "\mathcal{POR}\text{ closure}"] & \iota_{g_1} \arrow[u, "\text{Representability}"]
\end{tikzcd}$$

Figure 2 Summary of the proof of the existence of G^v

so, we introduce a set of basic predicates, which will be employed for the definition of G:

$$\begin{aligned}
\mathit{Circ}(x) &:= x \text{ is the encoding of a circuit with a single output}\\
\mathit{Eval}(x,k,y,t) &:= \text{When fed with inputs encoded by } y,\ x \text{ produces in output } t \text{ modulo } k\\
\mathit{NumVar}(x,n) &:= x \text{ is the encoding of an arithmetic circuit with } n \text{ variables}\\
\mathit{Degree}(x,d) &:= \text{The degree of the arithmetic circuit encoded by } x \text{ is } d\\
T(x,y) &:= \bigvee_{\overline{x}\in\bigcup_{i=0}^{\varrho}\{0,1\}^i} (x=\overline{x}\wedge y=y_{\overline{x}})\\
K(r,z) &:= z \text{ is a uniformly chosen random string in } \{0,1\}^{|r|}.
\end{aligned}$$

All these predicates characterize polytime random functions; for this reason, we can assume without lack of generality that they are Σ^b_1-formulæ of $\mathcal{RL}$. Using some of these predicates, it is possible to define a formula G_1 which executes one evaluation of the polynomial x:

$$\begin{aligned}
&G_1(x,m,n,d,z,y) := [m\leq\varrho\wedge T(x,1)\to y=0\wedge T(x,0)\to y=1]\vee\Big[(m>\varrho)\wedge\\
&\big((y=0\wedge\exists z_0,z_1.|z_0|=2m\wedge|z_1|=n\cdot(m+3)\wedge z_0\cdot z_1=z\wedge\exists t\preceq z.\,(\mathit{Eval}(x,z_0,z_1,t)\wedge t\neq 0))\vee\\
&(y=1\wedge\exists z_0,z_1.|z_0|=2m\wedge|z_1|=n\cdot(m+3)\wedge z_0\cdot z_1=z\wedge\mathit{Eval}(x,z_0,z_1,0))\big)\Big].
\end{aligned}$$

▶ Remark 33. The formula G_1 is a Σ^b_1 predicate of $\mathcal{RL}$ which characterizes one iteration of the algorithm `PZT`.

Differently from the original algorithm, the formula G_1 employs an additional parameter z as a *source of randomness* to determine the values of $r_i,\dots,r_n$ and k. This way, we are able to isolate the randomization part in a small portion of the formula we are building, i.e. that one where we determine the value of z by means of the predicate `Flip`.

Thanks to this construction, G_1 is a Σ^b_1 formula realizing some `Flip`-less $\mathcal{POR}$ function g_1. We can leverage this fact to define another `Flip`-less function $\iota_{g_1}(x,n,d,y,z,i)$ which iterates the function g_1 i times with the i-th $(n(m+3)+2m)$-long sub-string of z as source of randomness — i.e. as argument for g's z — and returns ϵ if and only if all these executions of g returned 1, otherwise it returns 0.

This proves that there is a `Flip`-less Σ^b_1 formula G^v realizing that function provable under $R\Sigma^b_1$-NIA — and even S^0_1. A picture of the proof of the existence of G^v is given in Figure 2. Intuitively, the formula G_v characterizes steps 2-7 of the `PZT` algorithm.

In order to define G, we will only need to compose G_v with another sub-formula which characterizes the first step of the `PZT` algorithm. This is quite simple because we only need to encode the generation of the values of d,n,m,z and to fix a number of iterations, as we will show in section C.2, $37m$ is an appropriate choice. Thus, the Σ^b_1 formula G can be defined as follows:

$$G(x,y) := \exists m \preceq x.|x| = m \wedge \Big[\big((Circ(x) \wedge \exists n \preceq 2^m.\exists d \preceq 2^m.\exists z.|z| = 37m \cdot (n(m+3)+2m) \wedge$$
$$NumVar(x,n) \wedge Degree(x,d) \wedge K(1^{37m\cdot(n(m+3)+2m)}, z) \wedge G^v(x,m,n,d,y,z,37m)\big) \vee \big(\neg Circ(x) \wedge y = 0\big)\Big].$$

C.2 Proving the Error Bound

Within this section, we argument that:

$$\mathsf{PA} \vdash \forall x.\forall y.\mathsf{TwoThirds}[G(x,y) \leftrightarrow H(x,y)],$$

which is equivalent to showing that:

$$\mathsf{PA} \vdash \forall x.\forall y.\mathsf{Threshold}[\mathsf{NoFlip}[G](x,y,z) \leftrightarrow H(x,y)].$$

In turn, it is possible to obtain a formula equivalent to $\mathsf{NoFlip}[G](x,y,z)$ by removing the quantification over z and the randomization its value by means of K. Doing so, we are able to obtain an equivalent claim:

$$\mathsf{PA} \vdash \forall x.\forall y.\exists m,d,n \preceq 1^{|x|}.|x| = m \wedge NumVar(x,n) \wedge Degree(x,d)$$
$$\exists^{\frac{2}{3}2^{37m\cdot(n(m+3)+2m)}} z.|z| = 37m \cdot (n(m+3)+2m) \wedge (G^v(x,n,d,z,y) \leftrightarrow H(x,y)).$$

It also is easy to observe $\exists^{\geq h} z.|z| = k \wedge P(z) \Leftrightarrow |\{z \in \mathbb{B}^k \mid P(z)\}| \geq h$. This allows to replace the threshold quantification with a formula of PA measuring the cardinality of some finite set. Therefore, we can reduce out goal to showing Lemma 34 using only the instruments provided by PA:

▶ **Lemma 34.** *For every encoding of a polynomial circuit x, for every $y \in \mathbb{B}$, whereas x has n variables, size m and degree d, it holds that:*

$$|\{z \in \mathbb{B}^{37m\cdot(n(m+3)+2m)} \mid (G^v(x,n,d,z,y) \leftrightarrow H(x,y))\}| \geq \frac{2}{3} \cdot 2^{37m\cdot(n(m+3)+2m)}.$$

Since G and H characterize two decisional functions, the claim of Lemma 34 can be restated in the following way:

$$|\{z \in \mathbb{B}^{37m\cdot(n(m+3)+2m)} \mid (G^v(x,n,d,z,0) \wedge H(x,0)) \vee (G^v(x,n,d,z,\epsilon) \wedge H(x,\epsilon))\}| \geq \frac{2}{3} \cdot 2^{37m\cdot(n(m+3)+2m)}.$$

Thus, it is possible Lemma 34 as a consequence of (1) and (2) which, respectively, can be stated as Lemmas 35 and 36.

▶ **Lemma 35** (Claim (1)). *For every $z \in \mathbb{B}^{37m\cdot(n(m+3)+2m)}$, every encoding of a polynomial circuit x, every $y \in \mathbb{B}$, whereas x has n variables, size m and degree d, it holds that $G^v(x,n,d,z,0) \to H(x,0)$.*

Proof. This result is a consequence of the compatibility of mod$k$ with respect to addition, multiplication ad inverse in $\mathbb{Z}$. Precisely: for every $n,m,k \in \mathbb{Z}$ with $k \geq 2$, it holds that:

1. $(n \odot m) \mod k = ((n \mod k) \odot (m \mod k)) \mod k$ for $\odot \in \{+,\cdot\}$.
2. $(-n) \mod k = -(n \mod k)$.
3. $n \mod k \neq 0 \to n \neq 0$.

These claims are shown as follows:

1. In this case, we will only show the case for $+$, that of $\cdot$ is analogous. The proof goes by induction on the recursion parameter, e.g. x. The case $x = 0$ is trivial. For the inductive case, let $x = ak + b$ and $y = ck + d$ for $b,d < k$, the existence and uniqueness of this

decomposition can be shown by induction on x. The IH tells that $(x \mod k + y \mod k) \mod k = b + d \mod k$.

$$\begin{aligned}(x+1 \mod k + y \mod k) \mod k &= (x+1 \mod k + d) \mod k \\ &= (b+1 \mod k + d) \mod k\end{aligned}$$

Now, if $b < k-1$, then $(b+1 \mod k+d) \mod k = b+1+d \mod k = ak+b+1+ck+d \mod k = x+1+y \mod k$. If $b = k-1$

$$\begin{aligned}(x+1 \mod k + y \mod k) \mod k &= (d) \mod k \\ &= (b+1+ak+ck+d) \mod k \\ &= x+1+y \mod k\end{aligned}$$

2. This proof goes by cases on n. The case where $n = 0$ is trivial. The case $n+1$ relies on the uniqueness of the decomposition of $n = ak + b$, which allows us to show that $-((ak+b+1) \mod k) = -((b+1) \mod k)$. If $0 \leq b < k-1$, then it is equal to $-b-1$, otherwise it is equal to 0. On the other hand, $(-ak-b-1) \mod k = (-b-1) \mod k$ and still, if $0 \leq b < k-1$, then this is equal to $-b-1$, otherwise it is equal to 0.

3. The counter-nominal is trivial: $n = 0 \to n \mod k = 0$.

Leveraging points 1 and 2, we show that the evaluation of a polynomial x as performed by the predicate *Eval* on input $\vec{r}$ is equal to $x(\vec{r}) \mod k$, the assumption $G^v(x, n, d, \vec{r}k, 0)$ allows us to conclude the premise of point 3, thus an application of point 3, allows us to conclude that $\mathbb{Z} \models p(\vec{x}) \neq 0$, by the definition of H, we conclude that $H(x, 0)$ holds. ◀

▶ **Lemma 36** (Claim (2)). *For every encoding of a polynomial circuit x, every $y \in \mathbb{B}$, whereas x has n variables, size m and degree d, it holds that:*

$$|\{z \in \mathbb{B}^{37m\cdot(n(m+3)+2m)} \mid G^v(x, n, d, z, \epsilon) \to H(x, \epsilon)\}| \geq \frac{2}{3} \cdot 2^{37m\cdot(n(m+3)+2m)}.$$

C.3 Proof of Lemma 36

Lemma 36 is shown finding an upper bound to the probability of error of the algorithm, i.e. to the number of values for z causing $G^v(x, n, d, z, \epsilon) \to H(x, \epsilon)$. Intuitively, there are two possible causes of error within the algorithm underlying the formula G.

1. if the outcome of the evaluation of the polynomial on $\vec{r}$ is some value $y \neq 0$ and k divides y, then the algorithm will reject its input even though x belongs to the language.

2. if the values $\vec{r}$ are a solution of the *non-identically zero* polynomial, then the algorithm will reject x, even though it belongs to the language.

The bound to the first error is found in section C.3.1, while a bound to the probability of the second error is found in section C.3.2. Lemma 36 can be shown combining these results.

C.3.1 Estimation of the Error, Case 1

▶ **Proposition 11** (Argument in [2]). There is some $\varrho \in \mathbb{N}$ such that for every $m \in \mathbb{N}$ greater than ϱ, every $0 \leq y < 2^{(m+3)\cdot 2^m}$:

$$|\{k \in \{1, \ldots, 2^{2m}\} \mid\} \mid k \text{ is not } \textit{a prime not dividing } y\}| \leq 2^{2m} - \frac{2^{2m}}{16m}$$

This probability depends on the range bounding k and on the number of evaluations taken in exam. The proof of this result relies on the following observations:

(a) Thanks to the Prime Number Theorem, there is some m' such that, for every $m \geq m'$, the number of primes in $\{1, \ldots, 2^{2m}\}$ is at least $K = \frac{2^{2m}}{4 \cdot 2m}$.
(b) For sufficiently big m, the number of primes in $\mathbb{Z}$ dividing $0 \leq y < 2^{(m+3) \cdot 2^m}$ is smaller than $L = 3m \cdot 2^m$.
(c) For every $n \in \mathbb{N}$, it holds that $3(2n + 201) \leq 2^{n+96}$.
(d) For every $m \geq 100$, said L the number of primes dividing y, it holds that $L \leq \frac{K}{2} = \frac{2^{2m}}{16m}$. This is shown by induction using (c).

As we anticipated, points (a), (b), (d) hold only for sufficiently big values of m. For this reason, we define ϱ as the greatest of these values.

C.3.1.1 Point (a)

We exploit here the fact, proved in [15], that the PNT is provable in PA (actually, in the fragment $I\Delta_0 + \mathsf{Exp}$); the formulation of the PNT can be paraphrased as follows:

$$\forall x \in \mathbb{N}. \forall q \in \mathbb{Q}^+. \exists a_x, z_x. z_q \leq x \Rightarrow \left| \frac{\pi(x) \log^*_{a_x}(x)}{x} - 1 \right| \leq q$$

Where $\pi(x)$ is the number of primes in $\{1, \ldots, x\}$ and $\log^*_a(x)$ is a *good approximation* of $\log_2(x)$, i.e. $\forall x. \log_2(x) \leq 2 \log^*_{a_x}(x) \leq 4 \log_2(x)$. This can be used to deduce that:

$$z_q \leq x \Rightarrow \frac{x}{\log^*_{a_x}(x)} (1 - q) \leq \pi(x) \leq \frac{x}{\log^*_{a_x}(x)} (1 + q),$$

Thus, fixing $q = \frac{1}{2}$ we obtain the following claim:

$$z_q \leq x \Rightarrow \frac{x}{4 \cdot \log_2(x)} \leq \frac{x}{2 \log^*_{a_x}(x)} \leq \pi(x)$$

We have shown that:

$$\forall x. \pi(x) = |\{p \in \{1, \ldots, x\} \mid p \text{ is prime}\}| \geq \frac{x}{4 \cdot \log_2(x)}.$$

For sake of readability, we name $\Pi(x)$ the set $\{p \in \{1, \ldots, x\} \mid p \text{ is prime}\}$.

C.3.1.2 Point(b)

To this aim, it suffices to observe that y is the product of $q_1, \ldots, q_L$ primes where $L \leq |y| = \log(2^{(m+3)2^m})$. This is a consequence of the Fundamental Theorem of Arithmetic.

▶ **Theorem 37** (The Fundamental Theorem of Arithmetic $\in$ PA)**.** *$\forall n \geq 2. prime(n) \vee \exists S, s. card(S, s) \wedge s \leq |n| \wedge \forall q \in S. n|q$ is provable in* PA.

Proof. We observe that the predicates $\in, |, prime, card$ can be easily modeled in PA. Then we go by induction on n to show the claim. The base case is trivial since 2 is prime and can be divided only by 2, so the cardinality of S is smaller than $|2|$. For the inductive case suppose that $n + 1$ is prime, in this case the claim is trivial, otherwise, if $n + 1$ is not prime, there are $a, b \in \mathbb{N}$, $ab = n + 1$, thus we can apply the IH on a, b, building S as the union of S_a, S_b. ◀

Thus, $L \leq |y| = (3 + m) \cdot 2^m = 3 \cdot 2^m + m \cdot 2^m \leq 3m \cdot 2^m$ — the last step is for $m \geq 1$, and is shown by induction. To sum up:

$$\forall 0 \leq y \leq 2^{(m+3)2^m}. |\{p \in \mathbb{Z} \mid q \text{ divides } y\}| \leq 3m \cdot 2^m.$$

For simplicity's sake, we omit the proof of (c) and (d), which are standard by induction.

Proof of Proposition 11. From (d) we can deduce that $|\{p \in \Pi(2^{2m}) \mid p \text{ does not divide } y\}| > \frac{2^{2m}}{4 \cdot 2m}$. Thus, the claim is a trivial consequence of the following observation, which concludes the proof:

$$|\{1, \ldots, 2^{2m} - 1\} \setminus \{p \in \Pi(2^{2m}) \mid p \text{ does not divide } y\}| \leq \\ 2^{2m} - |\{p \in \Pi(2^{2m}) \mid p \text{ does not divide } y\}|$$

◀

C.3.2 Estimation of Error, case 2

The goal of this section is to show that for every non-zero n-variate polynomial p of degree d and every set $S \subseteq \mathbb{Z}$, S contains sufficiently many witnesses of the fact that p is a non-zero polynomial.

▶ **Lemma 38** (Schwartz-Zippel). *For every n-*variate *polynomial p, for every $S \subseteq \mathbb{Z}$, said d the degree of p, it holds that:*

$$|\{(x_1, \ldots, x_n) \in S^n \mid \mathbb{Z} \models p(x_1, \ldots, x_n) = 0\}| \leq d|S|^{n-1}$$

The proof of this result is by induction on the number of variables and the degree of the polynomial p. It relies on a weak statement of the Fundamental Theorem of Algebra, stating that each univariate polynomial has at most d roots in $\mathbb{Z}$, where d is the degree of the polynomial. This is shown in Lemma 41. We start by defining the notion of non-zero univariate polynomial:

▶ **Definition 36.** We say that $p \in \mathsf{POLY}$ is a univariate *irreducible* polynomial if and only if it is univariate, and $\mathbb{Z} \models \forall x.p(x) \neq 0$. Moreover, we say that $p \in \mathsf{POLY}$ is a *non-zero* polynomial if and only if $\mathbb{Z} \models \exists x.p(x) \neq 0$.

This notion extends naturally to multivariate polynomials. The first step of the proof is to show that each univariate polynomial in $\mathbb{Z}$ can be expressed as the product of $\prod_{i=0}^{i<k}(x-\overline{x}_i)q(x)$ for k smaller than the degree of d. This result relies itself on the proof of the correctness of the polynomial division algorithm, in particular on the following properties:

▶ Remark 39.

1. Let p be an univariate polynomial in normal form with coefficients in $\mathbb{Z}$ and let $\overline{x}$ be a solution of p, The division algorithm applied on $p, (x - \overline{x})$ outputs a polynomial q with no remainder.
2. Let p be an univariate polynomial in normal form with coefficients in $\mathbb{Z}$ and let r be a non-zero polynomial in normal form with degree smaller than that of p. If q is obtained with remainder 0 applying the division algorithm to p and q, it holds that $rq = p$.
3. The degree of $r = p/q$ plus the degree of q is equal to the degree of p.

Although these results may seem very simple, for our purpose it is important to ensure that it can be shown in PA as well. This is true, because the algorithm performing the polynomial division is total and can be defined by induction on the number of monomials in the normal form of the dividend, thus its correctness can be proved by induction on this value as well.

Now, we want to prove that each uni-variate non-zero polynomial of degree d has at most d roots in $\mathbb{Z}$. We start by showing that it is always possible to express it as a product of simpler polynomials.

▶ **Lemma 40.** *For every univariate polynomial p there is a polynomial t:*

$$t(x) := \prod_{i=0}^{i<k}(x-\overline{x}_i)q(x)$$

with q an irreducible polynomial of degree equal to deg(p) − k, such that:

$$\mathbb{Z} \models \forall x.p(x) = t(x).$$

Proof. The formula we want to show is:

$$\forall p.\forall d \leq t(p).NumVar(p,1) \wedge Degree(p,d) \rightarrow \exists k < d.\exists q, x_0, \ldots, x_{k-1}.$$
$$\forall x.\prod_{i=0}^{i<k}(x-\overline{x}_i)q(x) = p(x) \wedge \forall x.q(x) \neq 0.$$

By induction on d:

- If it is equal to zero, then the normal form of p is a constant, so it suffices to take p as a candidate for q.
- If it is not equal to zero, we have two cases: if p is non-zero, then we can take p as a candidate for q, otherwise, we know that there is an $\overline{x} \in \mathbb{Z}$ such that $\mathbb{Z} \models p(\overline{x}) = 0$. Let p' be the polynomial obtained dividing p for $(x - \overline{x})$, then it holds that the degree of p' is equal to $deg(p) - 1$, so we can apply the induction hypothesis on p' and obtain that $\mathbb{Z} \models \forall x.p'(x) = \prod_{i=0}^{i<k-1}(x-\overline{x}_i)q(x)$, so $\mathbb{Z} \models \forall x.p(x) = (x-\overline{x})\prod_{i=0}^{i<k-1}(x-\overline{x}_i)q(x)$.

◀

▶ **Lemma 41.** *For each non-zero univariate polynomial it holds that:*

$$|\{x \in \mathbb{Z} \mid \mathbb{Z} \models p(x) = 0\}| \leq deg(p)$$

Proof. This result is shown applying the previous lemma:

$$|\{x \in \mathbb{Z} \mid \mathbb{Z} \models p(x) = 0\}| = |\{x \in \mathbb{Z} \mid \mathbb{Z} \models \prod_{i=0}^{i<k}(x-\overline{x}_i)q(x) = 0\}| =$$
$$= |\{x \in \mathbb{Z} \mid \mathbb{Z} \models \prod_{i=0}^{i<k}(x-\overline{x}_i) = 0\}| = k \leq deg(p)$$

◀

▶ **Corollary 12.** For every $S \subseteq \mathbb{Z}$ and for every non-zero univariate polynomial, it holds that:

$$|\{x \in S \mid \mathbb{Z} \models p(x) = 0\}| \leq deg(p).$$

Proof of Lemma 38. The proof is by induction on n:

- The claim for $n = 1$ coincides with Corollary 12.
- In the general case, we take the normal form of p, call that polynomial p'. It holds that

  $$\mathbb{Z} \models \forall \vec{x}.p(\vec{x}) = p'(\vec{x}),$$

  then we go by induction on d, we can factorize x_1 from each monomial of p and show that

  $$\exists k.\mathbb{Z} \models \forall x_1, \ldots, x_n.p'(x_1, \ldots, x_n) = \sum_{i=0}^{k} x_1^i \cdot p_i(x_2, \ldots, x_n)$$

Where p_k is non-zero. Applying the IH on d to p_k, we obtain that:

$$\{(x_2,\dots,x_n)\in S^{n-1} \mid \mathbb{Z}\models p_k(x_2,\dots,x_n)=0\} \leq (d-k)|S|^{n-2}$$

Fix some $(y_2,\dots,y_n)\in S^{n-1}$ and assume that $p_k(y_2,\dots,y_n)\neq 0$. In this case, the polynomial $\overline{p}_{y_2,\dots,y_n}(x) := \sum_{i=0}^{k} x^i \cdot p_i(y_2,\dots,y_n)$ is not identically zero because it is a normal form and a normal form is identically zero if and only if all its coefficients are different from zero, but the coefficient of x^k is different from 0 for construction. Thus, we can apply the IH on n, showing that:

$$\forall(y_2,\dots,y_n)\in S^{n-1}.|\{x\in S \mid \mathbb{Z}\models \overline{p}_{y_2,\dots,y_n}(x)=0\}|\leq k, \tag{$*$}$$

We continue by observing that

$$\forall(y_2,\dots,y_n)\in S^{n-1}.\{x\in S \mid \mathbb{Z}\models \overline{p}_{y_2,\dots,y_n}(x)=0\} = \{x\in S \mid \mathbb{Z}\models p(x,y_2,\dots,y_n)=0\},$$

for the definition of $\overline{p}$. We can conclude that

$$\begin{gathered}
\{(x_1,\dots,x_n)\in S^n \mid \mathbb{Z}\models p(x_1,\dots,x_n)=0\} = \\
\{(x_1,\dots,x_n)\in S^n \mid \mathbb{Z}\models p(x_1,\dots,x_n)=0 \land p_k(x_2,\dots,x_n)=0\}\cup \\
\{(x_1,\dots,x_n)\in S^n \mid \mathbb{Z}\models p(x_1,\dots,x_n)=0 \land p_k(x_2,\dots,x_n)\neq 0\} = \\
\{(x_1,\dots,x_n)\in S^n \mid \mathbb{Z}\models p(x_1,\dots,x_n)=0 \land p_k(x_2,\dots,x_n)=0\}\cup \\
\{(x_1,\dots,x_n)\in S^n \mid \mathbb{Z}\models p(x_1,\dots,x_n)=0 \land p_k(x_2,\dots,x_n)\neq 0 \land \overline{p}_{x_2,\dots,x_n}(x_1)=0\} \subseteq \\
\{(x_1,\dots,x_n)\in S^n \mid \mathbb{Z}\models p_k(x_2,\dots,x_n)=0\}\cup\{(x_1,\dots,x_n)\in S^n \mid \mathbb{Z}\models \overline{p}_{x_2,\dots,x_n}(x_1)=0\} = \\
S\times\{(x_2,\dots,x_n)\in S^{n-1} \mid \mathbb{Z}\models \overline{p}_{x_2,\dots,x_n}(x_1)=0\}\cup\{(x_1,\dots,x_n)\in S^n \mid \mathbb{Z}\models \overline{p}_{x_2,\dots,x_n}(x_1)=0\}
\end{gathered}$$

This result can be lifted to sizes, obtaining the claim.

◀

Proof of Lemma 36. We omit the case for $|x|<\varrho$, since the conclusion is trivial. The claim is equivalent to the following one:

$$\left|\left\{z=\begin{pmatrix} x_{1,1} & \dots & x_{n,1} & k_1\\ x_{1,2} & \dots & x_{n,2} & k_2\\ \vdots & \vdots & \ddots & \vdots\\ x_{1,37m} & \dots & x_{n,37m} & k_r\end{pmatrix}\in\left(\{0,2^{m+3}-1\}^n\times\{1,2^{2m}\}\right)^{37m} \mid G^v(p,m,n,d,\epsilon,z,37m)\land H(p,0)\right\}\right|\leq \tfrac{1}{3}2^{37m\cdot(n(m+3)+2m)}$$

The set on the left is:

$$\left\{\begin{pmatrix} x_{1,1} & \dots & x_{n,1} & k_1\\ x_{1,2} & \dots & x_{n,2} & k_2\\ \vdots & \vdots & \ddots & \vdots\\ x_{1,37m} & \dots & x_{n,37m} & k_r\end{pmatrix}\in\left(\{0,2^{m+3}-1\}^n\times\{1,2^{2m}\}\right)^{37m} \mid \forall 1\leq j\leq 48m.\mathbb{Z}\models p(\vec{x}_j)=0\lor(\mathbb{Z}\models p(\vec{x}_j)\neq 0\land k_j \text{ does divide } p(\vec{x}_j))\right\}$$

Thus, it is equal to:

$$\begin{aligned}
&\{(x_1,\dots,x_n,k)\in\left(\{0,2^{m+3}-1\}^n\times\{1,2^{2m}\}\right) \mid \\
&\qquad\qquad \mathbb{Z}\models p(\vec{x})=0\lor(\mathbb{Z}\models p(\vec{x})\neq 0\land k \text{ does divide } p(\vec{x}))\}^{37m}\subseteq
\end{aligned}$$

$$\begin{aligned}
&\big(\{(x_1,\dots,x_n,k)\in\{0,2^{m+3}-1\}^n\times\{1,2^{2m}\} \mid \mathbb{Z}\models p(\vec{x})=0\}\cup\\
&\{(x_1,\dots,x_n,k)\in\{0,2^{m+3}-1\}^n\times\{1,2^{2m}\} \mid \mathbb{Z}\models p(\vec{x})\neq 0\land k \text{ does divide } p(\vec{x})\}\big)^{37m}
\end{aligned}$$

Thus, its size is smaller than:

$$\big(|\{(x_1,\dots,x_n,k)\in\{0,2^{m+3}-1\}^n\times\{1,2^{2m}\}\mid\mathbb{Z}\models p(\vec{x})=0\}|+$$
$$|\{(x_1,\dots,x_n,k)\in\{0,2^{m+3}-1\}^n\times\{1,2^{2m}\}\mid\mathbb{Z}\models p(\vec{x})\neq 0\wedge k\text{ does divide }p(\vec{x})\}|\big)^{37m}.$$

Our goal is equivalent to showing the ratio between that value and $2^{37m\cdot(n(m+3)+2m)}$ is smaller than $\frac{1}{3}$. Which, in turn, resolves to:

$$\left(\frac{|\{(x_1,\dots,x_n,k)\in\{0,2^{m+3}-1\}^n\times\{1,2^{2m}\}\mid\mathbb{Z}\models p(\vec{x})=0\}|}{2^{n(m+3)+2m}}+\right.$$
$$\left.\frac{|\{(x_1,\dots,x_n,k)\in\{0,2^{m+3}-1\}^n\times\{1,2^{2m}\}\mid\mathbb{Z}\models p(\vec{x})\neq 0\wedge k\text{ does divide }p(\vec{x})\}|}{2^{n(m+3)+2m}}\right)^{37m}\leq\frac{1}{3}.$$

From Lemma 38 and point (e), it is smaller than:

$$\left(\frac{2^m\cdot 2^{(m+3)\cdot(n-1)}\cdot 2^{2m}}{2^{n(m+3)+2m}}+\right.$$
$$\left.\frac{|\{(x_1,\dots,x_n,k)\in\{0,2^{m+3}-1\}^n\times\{1,2^{2m}\}\mid\mathbb{Z}\models p(\vec{x})\neq 0\wedge k\text{ does divide }p(\vec{x})\}|}{2^{n(m+3)+2m}}\right)^{37m}=$$
$$\left(\frac{1}{8}-h+\frac{|\{(x_1,\dots,x_n,k)\in\{0,2^{m+3}-1\}^n\times\{1,2^{2m}\}\mid\mathbb{Z}\models p(\vec{x})\neq 0\wedge k\text{ does divide }p(\vec{x})\}|}{2^{n(m+3)+2m}}\right)^{37m}\leq$$

Applying Proposition 11, we can find an upper bound to this value, which is:

$$\left(\frac{1}{8}-h+\left(1-\frac{1}{8}+h\right)\frac{2^{n(m+3)+2m}-\frac{2^{n(m+3)+2m}}{16m}}{2^{n(m+3)+2m}}d\right)^{37m}=$$
$$\left(\frac{1}{8}-h+\left(\frac{7}{8}+h\right)\frac{2^{n(m+3)+2m}-\frac{2^{n(m+3)+2m}}{16m}}{2^{n(m+3)+2m}}\right)^{37m}=$$
$$\left(\frac{1}{8}-h+\left(\frac{7}{8}+h\right)\left(1-\frac{1}{16m}\right)\right)^{37m}=\left(\frac{1}{8}-h+\frac{7}{8}-\frac{7}{8\cdot 16m}+h-\frac{h}{16m}\right)^{37m}=$$
$$\left(1-\frac{7}{8\cdot 16m}-\frac{h}{16m}\right)^{37m}=\left(1-\frac{1}{16m}\left(\frac{7}{8}+h\right)\right)^{37m}\leq\left(1-\frac{1}{\frac{8}{7}16m}\right)^{37m}\leq$$
$$\leq\left(1-\frac{1}{\frac{8}{7}16m}\right)^{\frac{8}{7}16m}\cdot\left(1-\frac{1}{\frac{8}{7}16m}\right)^{\frac{8}{7}16m}\leq\frac{1}{4}$$

The very last observation is a consequence that $\forall n\geq 2.\left(1+\frac{1}{n}\right)^n\geq 2$. This, in turn, is done expanding the binomial $\left(1+\frac{1}{n}\right)^n$ and obtaining:

$$\left(1+\frac{1}{n}\right)^n=\sum_{k=0}^{n}\frac{n!}{(n-k)!k!}\frac{1}{n^k}=\sum_{k=0}^{n}\frac{1}{k!}\prod_{i=1}^{k-1}\left(1-\frac{i}{n^k}\right)=1+1+\sum_{k=2}^{n}\frac{1}{k!}\prod_{i=1}^{k-1}\left(1-\frac{i}{n^k}\right)$$

Where the derivation is justified by:

- The binomial expansion, namely: $\forall a,b,n.(a+b)^n=\sum_{k=0}^{n}\frac{n!}{k!(n-k)!}a^{n-k}b^k$. The proof is by induction on n. We will omit the details, which can be found in many text-books.
- $\forall k.k!\geq 2^k$. The proof is by induction. The base case is trivial, the inductive one is done as follows: $(k+1)!=k!(k+1)\geq 2^k(k+1)$. Now we go by cases on k: if it is 0, the claim is trivial, otherwise we observe that $(k+1)\geq 2$ and thus $2^k(k+1)\geq 2^{k+1}$.

- $\forall n. \sum_{k=1}^{n} 2^{-k} = 1 - 2^{-n}$, which is shown by induction on n. The base case is trivial. The inductive one is done as follows:$\sum_{k=1}^{n+1} 2^{-k} = \sum_{k=1}^{n} 2^{-k} + 2^{-n-1} = 1 - 2^{-n} + 2^{-n-1} = 1 - 2^{-n-1}$.

These proof are completely syntactic thus, modulo an encoding for rational numbers and for their operations, they can be done in PA without much effort. The proof proceeds observing that

$$\left(1 + \frac{1}{n}\right)^{n} = \left(1 + \frac{1}{n}\right)^{-1\cdot -1\cdot n} = \left(\frac{n}{n+1}\right)^{-n} = \left(1 - \frac{1}{n}\right)^{-n}.$$

Therefore, $\forall n \geq 4. \left(1 - \frac{1}{n}\right)^{n} \leq \frac{1}{2}$, and so:

$$\left(1 - \frac{1}{\frac{8}{7}16m}\right)^{\frac{8}{7}16m} \cdot \left(1 - \frac{1}{\frac{8}{7}16m}\right)^{\frac{8}{7}16m} \leq \frac{1}{4}.$$

◀

As we have shown, even last result can be proved in PA, using an encoding of finite sets and standard arithmetic on $\mathbb{N}$.

C.4 Closure of $\mathrm{BPP}_{\mathsf{PA}}$ under polytime reduction

The statement of Proposition 1 can be given more precisely as:

▶ **Proposition 12.** For every language $L \in \mathbf{BPP}_{\mathsf{PA}}$ and every language $M \in \mathbb{B}^*$, if there is an polytime function $r_{M,L} : \mathbb{S} \longrightarrow \mathbb{S}$, such that for every string in $\sigma \in \mathbb{S}$, $x \in M \leftrightarrow r(x) \in L$, then M is in $\mathbf{BPP}_{\mathsf{PA}}$.

Proof. Assume that $L \in \mathbf{BPP}_{\mathsf{PA}}$, and let G_L be the Σ_1^b $\mathcal{RL}$ formula characterizing L as required in Definition 9. Since $r_{M,L}$ is poly-time, there is a Σ_1^b `Flip`-less formula of $\mathcal{RL}$ R characterizing $r_{M,L}$ as well. This has a consequence that the formula $C(x,y) := \exists w \leq t(x).R(x,w) \wedge G_L(w,y)$ characterizes the composition of the reduction $r_{M,L}$ and the function χ_L. The function $C(x,y)$ is still a Σ_1^b formula, and characterizes a function deciding M due to the hypothesis on the correctness of the reduction $r_{M,L}$. Form this conclusion and $\mathrm{Lang}(G_L) = L$, we deduce $\mathrm{Lang}(C) = M$, we only need to show point (2) of Definition 9 for C:

$$\mathsf{PA} \vdash \forall x.\exists! y.\mathsf{TwoThirds}[C]$$
$$\mathsf{PA} \vdash \forall x.\exists! y.\mathsf{TwoThirds}[\exists w \leq t(x).R(x,w) \wedge G_L(w,y)]$$
$$\mathsf{PA} \vdash \forall x.\exists! y.\mathsf{Threshold}[\mathsf{NoFlip}[\exists w \leq t(x).R(x,w) \wedge G_L(w,y)]]$$
$$\mathsf{PA} \vdash \forall x.\exists w \leq t(x).R(x,w) \wedge \exists! y.\mathsf{Threshold}[\mathsf{NoFlip}[G_L(w,y)]]$$

This is a consequence of a Σ_1^b `Flip`-less formula of $\mathcal{RL}$ and the hypothesis on G_L. ◀