Enumerating Error Bounded Polytime Algorithms Through Arithmetical Theories

Melissa Antonelli, Ugo Dal Lago, Davide Davoli, Isabel Oitavem, Paolo Pistone

Enumerating Error Bounded Polytime Algorithms Through Arithmetical Theories

Melissa Antonelli
Helsinki Institute for Information Technology, Finland

Ugo Dal Lago
Bologna University, Italy
Inria, Université Côte d'Azur, Sophia Antipolis, France

Davide Davoli
Inria, Université Côte d'Azur, Sophia Antipolis, France

Isabel Oitavem
Center for Mathematics and Applications (NOVA Math), NOVA FCT, Caparica, Portugal
Department of Mathematics, NOVA FCT, Caparica, Portugal

Paolo Pistone
Bologna University, Italy

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: the first relies on measure-sensitive quantifiers, while the second is based on standard first-order quantification. This leads us to introduce a family of effectively enumerable subclasses of **BPP**, called $\mathbf{BPP}_{\mathrm{T}}$ and consisting of languages captured by those probabilistic Turing machines whose underlying error can be proved bounded in T. As a paradigmatic example of this approach, we establish that polynomial identity testing is in $\mathbf{BPP}_{\mathrm{T}}$, where $\mathrm{T} = \mathrm{I}\Delta_0 + \mathrm{Exp}$ is a well-studied theory based on bounded induction.

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

Digital Object Identifier 10.4230/LIPIcs.CSL.2024.10

Related Version *Extended Version*: `https://arxiv.org/abs/2311.15003` [1]

Funding The first, second, third and fifth authors' work is supported by the European Research Council through the project DIAPASoN ERC COoG 818616, and by the French "Agence Nationale de la Recherche" through the project PPS ANR-19-C48-0014. The first author's work is supported by the Helsinki Institute for Information Technology. The third author's work is supported by the French "Agence Nationale de la Recherche" through the project UCA DS4H ANR-17-EURE-0004. The fourth author's work is supported by national funds through the "FCT-Fundação para a Ciência e a Tecnologia, I.P.", through the projects UIDB/00297/2020 and UIDP/00297/2020 (Center for Mathematics and Applications).

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 [55] and Church [9]). 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 [17, 38, 54]), the theory of databases (e.g. through Codd's Theorem [11]) and computational complexity (e.g. through descriptive complexity [4, 39]). In particular, this last discipline deals with complexity classes [36, 10, 3], the nature of which still remains today, more than fifty years after the introduction of **P** and **NP** [12, 36], somewhat obscure.

The possibility of describing fundamental classes within the language of mathematical logic offered a better understanding of their nature: since the seventies [22, 15], but especially from the eighties and nineties [7, 32, 4, 39, 42], 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-of-recursion theoretical system 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 [37].

Among the various classes of problems considered in computational complexity, those defined on the basis of *randomized* algorithms [49] 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**. One might thus expect it to be possible to obtain an enumeration of **BPP**, along the lines of the many examples known for classes like **P**, or even **PP** [18, 19]. 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 of **BPP** relies on algorithms which are both efficient and not *too erratic*: once an input is fixed, one of the two possible outputs must clearly prevail over the other; in other words, there is some fixed probability p, bounded away from $\frac{1}{2}$, such that, on any input x, the machine outputs some value $b_x \in \{0, 1\}$ with probability at least p. The existence of an effective enumerable family of algorithms deciding *all and only* the problems in **BPP** is still an open question.

In this paper we make a step towards a logical understanding of semantic complexity classes, and in particular of the logical and proof-theoretic complexity involved in keeping error-bounds under control. Our contributions can be divided in three parts. First, we generalize to the probabilistic setting the path indicated by *bounded arithmetic* [7, 24], a well-known approach to capture polynomial time algorithms, by extending usual arithmetical languages with a distinguished unary predicate $\mathtt{Flip}(x)$, playing the role of a source of randomness. We define a bounded theory $R\Sigma^b_1$-NIA as the randomized analogue of Buss' S^1_2 [7] and Ferreira's Σ^b_1-NIA [25], and show that the functions which can be proved total in $R\Sigma^b_1$-NIA are precisely the polytime *random* functions [53], i.e. those functions from strings to *distributions* of strings which can be computed by polytime probabilistic Turing machines (PTM, for short). Then, we move towards proper randomized classes by considering ways to keep the probability of error under control *from within* the logic. We first consider *measure quantifiers* [48, 46, 2], well-studied second-order quantifiers capable of measuring *the extent* to which a formula is true; we then show that these quantifiers, when applied to bounded formulas, can be encoded via *standard* first-order quantification. This way we obtain two characterizations of the problems in **BPP**, yet still semantic in nature: the error-bound 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.

While these results, which rely on semantic conditions, do not shed light on the enumeration problem for **BPP** directly, they set the conditions for a proof-theoretic investigation of this class: our last contribution is the introduction of a family of new *syntactic* subclasses of

BPP, each called $\mathbf{BPP}_{\mathrm{T}}$, and consisting of those languages for which the error-bounding condition is not only true, but also *provable* in some (non necessarily bounded) theory T. This reduces the enumeration problem to that of finding a recursively enumerable (r.e., for short) arithmetical theory T such that $\mathbf{BPP} = \mathbf{BPP}_{\mathrm{T}}$. To witness the difficulty of this problem, we show that the error-bounding condition is Π_1^0-*complete* and that establishing that **BPP** *cannot* be enumerated would be at least *as hard* as refuting the $\mathbf{BPP} = \mathbf{P}$ conjecture. At the same time, we show that *polynomial identity testing* (PIT), one of the few problems in **BPP** currently not known to be in **P** lies in $\mathbf{BPP}_{\mathrm{T}}$, where $\mathrm{T} = \mathrm{I}\Delta_0 + \mathrm{Exp}$ is a well-studied [35] sub-theory of PA, thus identifying an interesting and effectively enumerable subclass of **BPP**.

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

- We introduce the arithmetical theory $R\Sigma_1^b$-NIA and prove that the random functions which are Σ_1^b-representable in it are precisely those 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 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 the extended version of this paper [1].
- We exploit this 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. This is in Section 4.
- Finally, we introduce a family of syntactic subclasses $\mathbf{BPP}_{\mathrm{T}} \subseteq \mathbf{BPP}$ of *provable* **BPP**-problems, relative to a theory T. After showing that the property of being non-erratic is Π_1^0-complete, we establish that PIT is in $\mathbf{BPP}_{(\mathrm{I}\Delta_0+\mathrm{Exp})}$.We conclude by showing how our approach relates to existing works capturing **BPP** languages in bounded arithmetic [41]. All this can be found in Section 5 and Section 7.

Related Work. While a recursion-theoretic characterization of the syntactic class **PP** can be found in [18], most existing characterizations of **BPP** are based on some external, semantic condition [20, 47]. 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, [41] and [40] uses bounded arithmetic to provide characterizations of (both syntactic and semantic) randomized classes, such as **ZPP**, **RP** and **coRP**, and also provides a semantic characterization of **BPP**. An in-depth comparison is thus in order, and can be found in Section 7. Finally, [47] 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.

First of all, it is worth observing that, although the distinction between syntactic and semantic classes appears in many popular textbooks (e.g. in [3, 50]), in the literature these notions are not defined in a precise way. Roughly speaking, syntactic classes are those which can be defined via limitations on the *amount of resources* (i.e. units of either time or space) that the underlying algorithm is allowed to use. Typical examples are the class **P** of problems

solvable in polynomial time and the class **PSPACE** of problems solvable in polynomial space. Instead, the definition of a semantic class usually requires, beyond some resource condition, an additional condition, sometimes called a *promise*, typically expressing that the underlying algorithm returns the correct answer *often enough*. A typical example here is the class **BPP** considered in this paper (cf. Definition 12), corresponding to problems solvable in polynomial time by probabilistic algorithms with some fixed error bound strictly smaller than $\frac{1}{2}$. Sometimes the distinction between syntactic and semantic classes may be subtle. For instance, as we discuss in Section 4, the class **PP**, whose definition also comprises a promise, is generally considered a syntactic class.

Notice that the sense of the terms "syntactic" and "semantic", when referred to complexity classes, is not clearly related to the sense that these terms have in mathematical logic. To a certain extent, the analysis that we develop in this paper with the tools of bounded arithmetic may help to clarify this point. On the one hand, well-known results in bounded arithmetic (cf. [7, 8]) provide a characterization of syntactic classes like **P** in terms of purely *proof-theoretic* conditions (i.e. provability in some weak fragment of Peano Arithmetic); on the other hand, we establish that, for a semantic class like **BPP**, an arithmetical characterization can be obtained by employing *both* proof-theoretic and *model-theoretic* conditions (i.e. truth in the standard model of Peano Arithmetic).

A natural question is whether such genuinely semantic (i.e. model-theoretic) conditions can somehow be eliminated in favor of purely syntactic (i.e. proof-theoretic) ones. In fact, this is a non-trivial problem, since, as proved in Section 5 (cf. Proposition 21), the promise underlying **BPP** is expressed by a Π^0_1-*complete* arithmetical formula. One should of course recall, however, that the distinction between semantic and syntactic classes refers to *how* a class is defined and not to the underlying set of problems. It is thus of *intensional* nature. In other words, even if **P** and **BPP** are defined in a different way, it could well be that someday we discover that $\mathbf{P} = \mathbf{BPP}$: in this case **BPP** would become a syntactic class, and, as we show in Section 5 (cf. Proposition 20), a purely proof-theoretic characterization of **BPP** would be available.

The problem of showing that a complexity class can be enumerated (i.e. that one can devise a recursive enumeration of, say, Turing Machines solving all and only the problems in the class) provides a different, and useful, angle to look at the distinction between syntactic and semantic classes. Ordinary syntactic classes, such 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 [34]. 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 every algorithm in this class works in polytime. Many characterizations of **P** (and of other syntactic classes), as those based on safe-recursion [4, 45], light and soft linear logic [31, 30, 44], and bounded arithmetic [7], 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? Being resource bounded is not sufficient for an algorithm to solve a problem in some semantic class, since there can well be algorithms getting it wrong too often. For instance, it may well be that some probabilistic Turing Machine running in

polynomial time does not solve *any* problem in **BPP**. For this reason, unfortunately, the enumeration strategy sketched above does not seem to be readily applicable to semantic classes. 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 the results in this paper, concerning proof-theoretic and model-theoretic characterizations of probabilistic complexity classes, may provide new insights on the nature of this problem, without giving a definite answer. Indeed, observe that the existence of a purely proof-theoretic characterization of some complexity class $\mathcal{C}$ via some recursively enumerable theory T directly leads to providing an enumeration of $\mathcal{C}$ (by enumerating the theorems of T). In this way, the problem of enumerating a semantic class $\mathcal{C}$ is directly related to the existence of some strong enough theory T.

In the following sections we do *not* prove **BPP** to be effectively enumerable, which is still out of reach. On the one hand we show that proving the non-enumerability of **BPP** is *as hard as* proving that **P** is different from **BPP**. On the other hand, we show that there exist subclasses of **BPP** which are large enough to include interesting problems in **BPP** and still "syntactic enough" to be effectively enumerable via some arithmetical theory.

3 Bounded Arithmetic and Polytime Random Functions

In this section we discuss our first result, namely, the characterization of polytime random functions via bounded arithmetic.

3.1 From Arithmetic to Randomized Computation, Subrecursively

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

Recursive Functions and Arithmetical Formulas. The study of so-called bounded theories of arithmetic, i.e. subsystems of PA in which only *bounded quantifications* are admitted, initiated by Parikh and Buss, has led to characterize several complexity classes [51, 14, 7, 8, 24, 43]. At the core of these characterizations lies the well-known fact (dating back to Gödel's [33]) that recursive functions can be *represented* in PA by means of Σ^0_1-formulas, i.e. formulas of the form $\exists x_1.\ldots.\exists x_n.A$, where A is a bounded formula. For example, the formula

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

represents the function $f(x_1, x_2) = (x_1 \times x_2) + 1$. 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 becomes possible to capture functions computable within given resource bounds [7, 8].

In order to extend this approach to classes of *random* computable functions, we rely on a simple correspondence between first-order predicates over natural numbers and *oracles* from the Cantor space $\{0,1\}^{\mathbb{N}}$, following [2]. Indeed, suppose the aforementioned recursive function f has now the ability to observe (part of) an infinite sequence of bits. For instance, f might observe the first bit and return $(x_1 \times x_2) + 1$ if this is 0, and return 0 otherwise. 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 $\texttt{Flip}(x)$, to be interpreted as a stream of (random) bits. Our function f can then be represented by the following formula:

$$B(x_1, x_2, y) := \big(\texttt{Flip}(\overline{0}) \wedge \exists x_3.x_1 \times x_2 = x_3 \wedge y = succ(x_3)\big) \vee \big(\neg\texttt{Flip}(\overline{0}) \wedge y = \overline{0}\big).$$

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 $\texttt{Flip}(x)$, the actual numerical function that B represents depends on the choice of an interpretation for $\texttt{Flip}(x)$, i.e. on the choice of an oracle for f.

In the rest of this section we develop this idea in detail, establishing a correspondence between polytime random functions and a class of *oracle-recursive* functions which are provably total in a suitable bounded theory relying on the predicate `Flip`.

The Language $\mathcal{RL}$. We let $\mathbb{B} := \{0, 1\}$, $\mathbb{S} := \mathbb{B}^*$ indicate the set of finite words from $\mathbb{B}$, and $\mathbb{O} := \mathbb{B}^{\mathbb{S}}$. We introduce a language for first-order arithmetic incorporating the new predicate symbol $\texttt{Flip}(x)$ and its interpretation in the standard model. Following [26], we consider a first-order signature for natural numbers *in binary notation*. Consistently, formulas will be interpreted over $\mathbb{S}$ rather than $\mathbb{N}$. Working with strings is not essential and all results below could be spelled out in a language for natural numbers. Indeed, bounded theories may be formulated in both ways equivalently, e.g. Ferreira's Σ^b_1-NIA and Buss' S^1_2 [26].

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

$$t, s ::= x \mid \epsilon \mid \texttt{0} \mid \texttt{1} \mid t \frown s \mid t \times s$$
$$F, G ::= \texttt{Flip}(t) \mid t = s \mid t \subseteq s \mid \neg F \mid F \wedge G \mid F \vee G \mid \exists x.F \mid \forall x.F.$$

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. As usual, we let $A \to B := \neg A \vee B$.

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

As for standard bounded arithmetics [7, 23], a defining feature of our theory is the focus on so-called *bounded quantification*. In $\mathcal{RL}$, *bounded quantifications* are of the forms $\forall x.\texttt{1}^x \subseteq \texttt{1}^t \to F$ and $\exists x.\texttt{1}^x \subseteq \texttt{1}^t \wedge F$, 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 Σ^b_1*-formula* if it is of the form $\exists x_1 \preceq t_1 \ldots . \exists x_n \preceq t_n.G$, 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 $\mathcal{O}(k^2)$, 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 the one introduced in [2]. 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 idea is that the predicate symbol `Flip` may stand for *an arbitrary* subset of $\mathbb{S}$, that is, an arbitrarily chosen $\omega \in \mathbb{O}$. For this reason, we take as the interpretation of a $\mathcal{RL}$-formula

F the set $[\![F]\!] \subseteq \mathbb{O}$ of *all* possible interpretations of `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* $\mathsf{C}^b_\sigma = \{\omega \mid \omega(\sigma) = b\}$, with $b \in \{0,1\}$, and μ is the *unique* measure such that $\mu(\mathsf{C}^b_\sigma) = \frac{1}{2}$ (see [5]). While the interpretation of terms is standard, the interpretation of 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:*

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

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

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, requires some ξ as above as well as 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 F is satisfied in the first-order environment formed by ξ *and* ω.

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 [25]. 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. As standard in bound arithmetic, 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.

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 PTCA [23, 25]. 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 stream of random bits. Furthermore, our class includes 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 $E(x, \omega) = \epsilon$, the *projection* functions $P^n_i(x_1, \ldots, x_n, \omega) = x_i$, the *word-successor* function $S_b(x, \omega) = xb$, the *conditional* function $C(\epsilon, y, z_0, z_1, \omega) = y$ and $C(xb, y, z_0, z_1, \omega) = z_b$, where $b \in \mathbb{B}$ (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

$$\begin{aligned} f(\vec{x}, \epsilon, \omega) &= g(\vec{x}, \omega) \\ f(\vec{x}, y0, \omega) &= h_0\big(\vec{x}, y, f(\vec{x}, y, \omega), \omega\big)|_{t(\vec{x},y)} \\ f(\vec{x}, y1, \omega) &= h_1\big(\vec{x}, y, f(\vec{x}, y, \omega), \omega\big)|_{t(\vec{x},y)}, \end{aligned}$$

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

We now show that functions of $\mathcal{POR}$ are precisely those which are Σ_1^b-representable in $R\Sigma_1^b$-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_1^b-representable in $R\Sigma_1^b$-NIA *if there exists a Σ_1^b-formula $G(\vec{x}, y)$ of $\mathcal{RL}$ such that:*

1. *$R\Sigma_1^b$-NIA $\vdash \forall \vec{x}.\exists! y.G(\vec{x}, y)$,*
2. *for all $\sigma_1, \ldots, \sigma_k, \tau \in \mathbb{S}$ and $\omega \in \mathbb{O}$, $f(\sigma_1, \ldots, \sigma_k, \omega) = \tau$ iff $\omega \in [\![G(\overline{\sigma_1}, \ldots \overline{\sigma_k}, \overline{\tau})]\!]$.*

Condition 1. above does *not* say that the unique value y is obtained as a function of $\vec{x}$ *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 this condition 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 $\mathcal{POR}$.

▶ **Theorem 4.** *For any $f : \mathbb{S}^k \times \mathbb{O} \to \mathbb{S}$, f is Σ_1^b-representable in $R\Sigma_1^b$-NIA iff $f \in \mathcal{POR}$.*

Proof sketch. ($\Leftarrow$) The desired Σ_1^b-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_1^b, 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 [51] (which holds for S_2^1 and Σ_1^b-NIA) to $R\Sigma_1^b$-NIA, to state that if $R\Sigma_1^b$-NIA $\vdash \forall \vec{x}.\exists y.G(\vec{x}, y)$, where $G(\vec{x}, y)$ is a Σ_1^b-formula, then we can find a term t such that $R\Sigma_1^b$-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$^\omega$ [13], and this goes through a *realizability interpretation* of the intuitionistic version of $R\Sigma_1^b$-NIA, called $IR\Sigma_1^b$-NIA. Further details can be found in the extended version of this paper [1]. ◀

3.2 Characterizing Polytime Random Functions

Theorem 4 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, like **BPP** or **RP**, which are defined in terms of functions computed by PTMs. 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 mean a function of the form $f : \mathbb{S}^k \to \mathbb{D}(\mathbb{S})$. Observe that any (polytime) PTM $\mathcal{M}$ computes a random function $f_{\mathcal{M}}$, where, for every $\sigma_1, \ldots, \sigma_k, \tau \in \mathbb{S}$, $f_{\mathcal{M}}(\sigma_1, \ldots, \sigma_k)(\tau)$ coincides with the probability that $\mathcal{M}(\sigma_1 \sharp \ldots \sharp \sigma_k) \Downarrow \tau$. However, a random function needs not be computed by a PTM in general. We define the following class of *polytime random functions*:

▶ **Definition 5** (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.*

Functions of **RFP** are closed under *monadic composition* $\diamond$, where $(g \diamond f)(\sigma)(\tau) = \sum_{\rho \in \mathbb{S}} g(\rho)(\tau) \cdot f(\sigma)(\rho)$ (one can check $f_{\mathcal{N}} \diamond f_{\mathcal{M}} = f_{\mathcal{N} \circ \mathcal{M}}$, 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_1^b-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 6.** *A function* $f : \mathbb{S}^k \to \mathbb{D}(\mathbb{S})$ *is* Σ_1^b-representable in $R\Sigma_1^b$-NIA *if there exists a* Σ_1^b*-formula* $G(\vec{x}, y)$ *of* $\mathcal{RL}$ *such that:*

1. $R\Sigma_1^b$*-NIA* $\vdash \forall \vec{x}.\exists! y.G(\vec{x}, y)$*,*
2. *for all* $\sigma_1, \ldots, \sigma_k, \tau \in \mathbb{S}$*,* $f(\sigma_1, \ldots, \sigma_k, \tau) = \mu\big([\![G(\overline{\sigma_1}, \ldots, \overline{\sigma}_k, \overline{\tau})]\!]\big)$*.*

Notice that any Σ_1^b-formula $G(\vec{x}, y)$ satisfying Condition 1. from Definition 6 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_1^b-represented by G. Moreover, if G represents some $f \in \mathbf{RFP}$, then $f = \langle G \rangle$. In analogy with Theorem 4, we can now prove the following result:

▶ **Theorem 7.** *For any* $f : \mathbb{S}^k \to \mathbb{D}(\mathbb{S})$*,* f *is* Σ_1^b*-representable in* $R\Sigma_1^b$*-NIA iff* $f \in \mathbf{RFP}$*.*

Thanks to Theorem 4, the proof of the result above simply consists in showing that $\mathcal{POR}$ and **RFP** can be related as stated below.

▶ **Lemma 8.** *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* $\sigma_1, \ldots, \sigma_k, \tau \in \mathbb{S}$*,* $\mu(\{\omega \mid f(\vec{\sigma}, \omega) = \tau\}) = g(\sigma_1, \ldots, \sigma_k, \tau)$*, and conversely.*

Proof sketch. The first step of our proof 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 $\sigma_1, \ldots, \sigma_k, \tau \in \mathbb{S}$, and $\eta \in \mathbb{B}^{\mathbb{N}}$, $f(\sigma_1, \ldots, \sigma_k, \tau) = \mu(\{\eta \mid g(\sigma_1, \ldots, \sigma_k, \eta) = \tau\})$, 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 [56], 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). ◀

4 Semantic Characterizations of BPP

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

▶ **Definition 9.** *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 $f(\sigma)(\epsilon) > \frac{1}{2}$.*

It is instructive to first take a look at the case of the class **PP**, recalled below:

▶ **Definition 10 (PP).** *Given a language $L \subseteq \mathbb{S}$, $L \in$ **PP** iff there is a polynomial time PTM $\mathcal{M}$ such that for any $\sigma \in \mathbb{S}$, $\Pr[\mathcal{M}(\sigma) = \chi_L(\sigma)] > \frac{1}{2}$, where, $\chi_L : \mathbb{S} \to \{0, 1\}$ is the characteristic function of L.*

At first glance, **PP** might be considered a semantic class, since its definition comprises *both* a resource condition and a promise. However, **PP** is generally considered a syntactic class, due to the fact that, when trying to capture the machines solving languages in **PP**, the promise condition can actually be eliminated. Indeed, *any* PTM $\mathcal{M}$ running in polynomial time recognizes *some* language in **PP**, namely the language $L = \mathrm{Lang}(f)$, where f is the polytime random function computed by $\mathcal{M}$. Furthermore, the class **PP** *can* be enumerated (see e.g. [18]).

Using Theorem 7, the remarks above readily lead to a proof-theoretic characterization of **PP** via $R\Sigma^b_1$-NIA.

▶ **Proposition 11** (Syntactic Characterization of **PP**). *For any language $L \subseteq \mathbb{S}$, $L \in$ **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 provides an enumeration of **PP** (by enumerating the pairs made of a formula G and a proof in $R\Sigma^b_1$-NIA of Condition 1). However, while a majority rule is enough to capture the problems in **PP**, the definition of a semantic class like **BPP** requires a different condition.

▶ **Definition 12 (BPP).** *Given a language $L \subseteq \mathbb{S}$, $L \in$ **BPP** iff there is a polynomial time PTM $\mathcal{M}$ such that for any $\sigma \in \mathbb{S}$, $\Pr[\mathcal{M}(\sigma) = \chi_L(\sigma)] \geq \frac{2}{3}$.*

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 13.** *A random function $f : \mathbb{S} \to \mathbb{D}(\mathbb{S})$ is* 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 14.** *For any language $L \subseteq \mathbb{S}$, $L \in$ **BPP** iff $L = \mathrm{Lang}(f)$, for some non-erratic random function $f \in$ **RFP**.*

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 $L \in$ **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 14 suggests that, in order to characterize **BPP** in the spirit of Proposition 11, a new condition has to be added, corresponding to the fact that G represents a non-erratic random function. 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 [2], then by purely arithmetical means. While both such methods ultimately consist in showing that the *truth* of a formula in the standard model of $R\Sigma_1^b$-NIA, they also suggest a more proof-theoretic approach, that we explore in Section 5.

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 [2], consists in enriching $\mathcal{RL}$ with *measure-quantifiers* [48, 46], 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 $[\![\mathbf{C}^{t/s}F]\!]_\xi = \mathbb{O}$ when $|[\![s]\!]_\xi| > 0$ and $\mu([\![F]\!]_\xi) \geq \frac{|[\![t]\!]_\xi|}{|[\![s]\!]_\xi|}$ both hold, and $[\![\mathbf{C}^{t/s}F]\!]_\xi = \emptyset$ otherwise. To improve readability, for all $n, m \in \mathbb{N}$, we abbreviate $\mathbf{C}^{\mathtt{1}^n/\mathtt{1}^m}F$ 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 15** (First 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. Let $L \in$ **BPP** and $g : \mathbb{S} \to \mathbb{D}(\mathbb{S})$ be a function of **RFP** computing L with uniform error-bound (which, thanks to Lemma 14, we can suppose to be non-erratic). By Theorem 7, there is a Σ_1^b-formula $G(x,y)$ such that $g = \langle G \rangle$. So, 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**. ◀

Arithmetizing Measure Quantifiers. Theorem 15 relies on the tight correspondence between arithmetic and probabilistic computation; yet, Condition 2. involves formulas which are not in the language of first-order arithmetic. Lemma 16 below shows that measure quantification over bounded formulas of $\mathcal{RL}$ can be expressed arithmetically.

▶ **Lemma 16** (De-Randomization of Bounded Formulas). *For any Σ_1^b-formula $F(\vec{x})$ of $\mathcal{RL}$, there exists a* `Flip`*-free Π_1^0-formula* $\mathsf{TwoThirds}[F](\vec{x})$ *such that for any $\vec{\sigma} \in \mathbb{S}$, $\vDash \mathsf{TwoThirds}[F](\overline{\vec{\sigma}})$ holds iff $\mu([\![F(\overline{\vec{\sigma}})]\!]) \geq \frac{2}{3}$.*

Proof. First, observe that for any bounded $\mathcal{RL}$-formula $F(\vec{x})$, strings $\vec{\sigma}$ 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 can construct a $\mathcal{RL}$-term $t_F(\vec{x})$ such that for any $\vec{\sigma} \in \mathbb{S}$ and $\omega, \omega' \in \mathbb{O}$, if ω and ω' agree on all strings shorter than $t_F(\vec{\sigma})$, then $\omega \in [\![F(\sigma)]\!]$ iff $\omega' \in [\![F(\sigma)]\!]$. Now, all finitely many relevant bits $\omega(\tau)$, for $|\tau| \leq t_F(\vec{\sigma})$ can be encoded as a *unique* string w of length $\leq 2^{|t_F(\vec{\sigma})|}$ where the bit w_i corresponds to the value $\omega(\tau)$, where τ is obtained by stripping the right-most bit from the binary representation of i. We obtain in this way a `Flip`-free formula $F^*(\vec{x}, y)$ such that measuring $[\![F(\sigma)]\!]$ corresponds to *counting* the strings y of length $\leq 2^{|t_F(\vec{\sigma})|}$ making $F^*(\vec{x}, y)$ true, i.e. to showing

$$\left| \left\{ \tau \preceq 2^{|t_F(\vec{\sigma})|} \;\middle|\; F^*(\vec{\sigma}, \tau) \right\} \right| \geq \frac{2}{3} \cdot N, \qquad (\star)$$

where $2^{\epsilon} = 1$ and $2^{\sigma b} = 2^{\sigma}2^{\sigma}$ is an exponential function on strings and $N = 2^{\left(2^{|t_F(\sigma)|}\right)}$ is the total amount of the strings to be counted. $(\star)$ can be encoded in a standard way yielding a bounded formula $F^{\sharp}(\vec{x})$ in the language of arithmetic *extended* with the function symbol 2^x. Finally, the function symbol 2^x can be eliminated using a Δ_0^0-formula $\exp(x, y)$ defining the exponential function (see [28]), yielding a `Flip`-free Π_1^0-formula of $\mathcal{RL}$ of the form $\forall z_1 \ldots . \forall z_k . \exp(t_1, z_1) \wedge \cdots \wedge \exp(t_k, z_k) \rightarrow F^{\sharp}(\vec{x}, z_1, \ldots, z_k)$. ◀

▶ Remark 17. It is important to observe at this point that the elimination of `Flip` via counting takes us *beyond* the usual machinery of bounded arithmetic, since we employ some operation which is not polytime. This is indeed not surprising, since the counting problems associated with polytime problems (generating the class $\sharp\mathsf{P}$) are not even known to belong to the polynomial hierarchy PH (while, by Toda's theorem, we know that $\mathsf{PH} \subseteq \mathsf{P}^{\sharp\mathsf{P}}$).

Theorem 15 and Lemma 16 together yield a purely arithmetical characterization of **BPP**. Let $\mathsf{NotErratic}[G]$ indicate the arithmetical formula $\forall x . \exists y \preceq 0 . \mathsf{TwoThirds}[G](x, y)$.

▶ **Theorem 18** (Second Characterization of **BPP**). *For any language $L \subseteq \mathbb{S}$, $L \in$ **BPP** when 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. *$\models \mathsf{NotErratic}[G]$,*
3. *$L = \mathrm{Lang}(\langle G \rangle)$.*

5 Provably BPP Problems

The characterization provided by Theorem 18 is still semantic in nature, as it provides no way to effectively enumerate **BPP**: the crucial Condition 2 is not checked within a formal system, but over the standard model of $\mathcal{RL}$. Yet, since the condition is now expressed in purely arithmetical terms, it makes sense to consider *syntactic* variants of Condition 2, where the model-theoretic check is replaced by provability in some sufficiently expressive theory.

We will work in extensions of $R\Sigma_1^b$-NIA+Exp, where $\mathrm{Exp} = \forall x . \exists y . \exp(x, y)$ is the formula expressing the totality of the exponential function (which is used in the de-randomization of Lemma 16). This naturally leads to the following definition:

▶ **Definition 19** (Class $\mathbf{BPP}_{\mathrm{T}}$). *Let $\mathrm{T} \supseteq R\Sigma_1^b$-NIA + Exp be a theory in the language $\mathcal{RL}$. The* class **BPP** relative to T, *denoted $\mathbf{BPP}_{\mathrm{T}}$, contains all languages $L \subseteq \mathbb{S}$ such that for some Σ_1^b-formula $G(x, y)$ the following hold:*

1. *$R\Sigma_1^b$-NIA $\vdash \forall x . \exists! y . G(x, y)$,*
2. *$\mathrm{T} \vdash \mathsf{NotErratic}[G]$,*
3. *$L = \mathrm{Lang}(\langle G \rangle)$.*

Whenever T is sound (i.e. $\mathrm{T} \vdash F$ implies that F is true in the standard model), it is clear that $\mathbf{BPP}_{\mathrm{T}} \subseteq \mathbf{BPP}$. However, a crucial difference between the *syntactic* class $\mathbf{BPP}_{\mathrm{T}}$ and the semantic class **BPP** is that, when T is recursively enumerable, $\mathbf{BPP}_{\mathrm{T}}$ can be *enumerated* (by enumerating the proofs of Condition 1. and 2. in T). Hence, the enumerability problem for **BPP** translates into the question whether one can find a sound r.e. theory T such that $\mathbf{BPP}_{\mathrm{T}} = \mathbf{BPP}$. Let us first observe that the relevance of this problem is tightly related to the question $\mathbf{BPP} = \mathbf{P}$:

▶ **Proposition 20.** *If $\mathbf{BPP} = \mathbf{P}$, then there exists a r.e. theory T such that $\mathbf{BPP} = \mathbf{BPP}_{\mathrm{T}}$.*

Proof. If $\mathbf{BPP} = \mathbf{P}$, and $L \in \mathbf{BPP}$, then there is a polytime deterministic TM μ accepting it. μ yields then a PTM μ^* in a trivial way. Since the corresponding formula G of $\mathcal{RL}$ does not contain `Flip`, $\mathsf{NotErratic}[G]$ can be proved in e.g. $R\Sigma_1^b$-NIA + Exp. ◀

The counter-positive of the result above is even more interesting, as it says that establishing that *no* r.e. theory T is such that $\mathbf{BPP}_{\mathrm{T}} = \mathbf{BPP}$ is *at least as hard as* establishing that $\mathbf{BPP} \neq \mathbf{P}$. Yet, without knowing whether $\mathbf{BPP} = \mathbf{P}$, how hard may it be to find a theory T such that $\mathbf{BPP} = \mathbf{BPP}_{\mathrm{T}}$?

Observe that, when G is Σ_1^b, $\mathsf{NotErratic}[G]$ is expressed by a Π_1^0-formula: as $\mathsf{TwoThirds}[G](\vec{x})$ is of the form $\forall \vec{z}.\, \wedge_i \exp(t_i, z_i) \to F^\sharp(\vec{x}, \vec{z})$, the condition is expressed by the Π_1^0-formula $\forall x. \forall \vec{z}.\, \wedge_i \exp(t_i, z_i) \to \exists y \preceq \mathsf{0}. F^\sharp(\vec{x}, \vec{z})$. Hence, if we us fix some recursive enumeration $(\mathcal{M}_n)_{n\in\mathbb{N}}$ of polytime PTM as well as a recursive coding $\sharp\mathcal{M}$ of such machines as natural numbers, the fact that $\mathcal{M}$ is non-erratic is expressed by some Π_1^0-formula $\varphi_{\mathsf{NotErratic}}(\sharp\mathcal{M})$. The Π_1^0-set $\mathsf{NotErratic} = \{e \mid \varphi_{\mathsf{NotErratic}}(e)\}$ indicates then the sets of codes corresponding to non-erratic machines.

The possibility of finding a theory strong enough to prove *all* positive instances of Condition 2 is then ruled out by the following result.

▶ **Proposition 21.** NotErratic *is* Π_1^0*-complete.*

Proof. We reduce to NotErratic the Π_1^0-complete problem HALT_{n^2} consisting of codes of TM halting in time at most n^2 (see [29]). With any TM μ associate a polytime PTM μ^* that, on input x, yields TRUE with prob. $\frac{1}{2}$, and otherwise simulates $\mu(x)$ on $|x|^2$ steps, yielding TRUE if the computation of $\mu(x)$ terminated, and FALSE otherwise. Then it is easily seen that $\mu \in \mathsf{HALT}_{n^2}$ iff $\mu^* \in \mathsf{NotErratic}$. ◀

▶ **Corollary 22.** *Codes of poly-time and non-erratic PTMs form a* Σ_2^0*-complete set.*

Proof. As we say, for a PTM, solving some **BPP**-problem is equivalent to being polytime and non-erratic. Being the code of a polytime (P)TM is a Σ_2^0-complete property [34]. By Proposition 21, checking non-erraticity does not increase the logical complexity. ◀

Proposition 21 implies that for any consistent theory T one can always find some non-erratic polytime PTM whose non-erraticity is *not provable* in T. Indeed, since NotErratic is Π_1^0-complete, we can reduce to it the Π_1^0-set of codes of *consistent* r.e. theories. Hence, if T is some consistent theory such that for *any* code $e \in \mathsf{NotErratic}$, T proves $\varphi_{\mathsf{NotErratic}}(e)$, then T can prove *all* Π_1^0-statement expressing the consistency of some consistent r.e. theory, and thus, in particular, the one expressing its own consistency, contradicting (Rosser's variant of) Gödel's second incompleteness theorem.

Observe that Corollary 22 suggests that the enumerability problem *might* be very difficult, but it *does not* provide a negative answer to it. Indeed, recall that what we are interested in is not an enumeration of *all* non-erratic polytime PTM, but an enumeration containing *at least one* machine for each problem in **BPP**. In other words, the question remains open whether, for any non-erratic polytime PTM, it is possible to find a machine solving the same problem but whose non-erratic behavior can be proved in some fixed theory T. While we do not know the answer to this question, we can still show that a relatively weak arithmetical theory is capable of proving the non-erraticity of a machine solving one of the (very few) problems in **BPP** which are currently not known to be in **P**.

6 Polynomial Zero Testing is Provably BPP

In this section we establish that PIT is in $\mathbf{BPP}_{(\mathrm{I}\Delta_0+\mathrm{Exp})}$. We recall that $\mathrm{I}\Delta_0 + \mathrm{Exp}$ is the fragment of Peano Arithmetics with induction restricted to *bounded formulas*, together with the totality of the exponential function.

▶ Remark 23. While $\mathrm{I}\Delta_0 + \mathrm{Exp}$ is a theory in the usual language of PA, here we work in a language for binary strings. Indeed, what we here call $\mathrm{I}\Delta_0 + \mathrm{Exp}$ is actually the corresponding theory Δ_0-NIA + Exp, formulated for the language $\mathcal{RL}$ *without* `Flip`, and defined as Σ_1^b-NIA + Exp with induction extended to *all* bounded formulas, plus the axiom Exp. Based on [26] Δ_0-NIA corresponds to Buss' theory S_2, which, in turn, is known to correspond to $\mathsf{I}\Delta_0 + \Omega_1$, indeed a sub-theory of $\mathrm{I}\Delta_0 + \mathrm{Exp}$.

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 $\mathcal{RL}$.

▶ **Definition 24** (cf. [3]). *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 $\mathbf{BPP}_{(\mathrm{I}\Delta_0+\mathrm{Exp})}$ is structured as follows:

- We identify a Σ_1^b-formula $G(x, y)$ of $\mathcal{RL}$ characterizing the polytime algorithm `PZT` from [3], and we turn it into a `Flip`-free formula $G^*(x, y, z)$ as in Lemma 16, where the variable z stands for the source of randomness;
- We identify a `Flip`-free Δ_0^0-formula $H(x, y)$ which represents the naïve deterministic algorithm for PZT.
- We show that $\mathrm{I}\Delta_0 + \mathrm{Exp}$ proves a statement showing that the formulas G^* and H are equivalent in at least $\frac{2}{3}$ of all (finitely many) relevant values of z. In other words, we establish $\mathrm{I}\Delta_0 + \mathrm{Exp} \vdash \forall x.\forall y.\mathsf{TwoThirds}[G(x, y) \leftrightarrow H(x, y)]$.

From the last step, since the totality of H is provable in $\mathrm{I}\Delta_0 + \mathrm{Exp}$, we can deduce $\mathrm{I}\Delta_0 + \mathrm{Exp} \vdash \forall x.\exists y.\mathsf{TwoThirds}[G](x, y)$, as required in Definition 19.

Each of the aforementioned steps will be described in one of the forthcoming paragraphs, although the details are discussed in the extended version of this paper [1].

The Randomized Algorithm. Our algorithm for PZT takes an input x, which encodes a circuit p of size m on the variables $v_1, \ldots, v_n$, it draws $r_1, \ldots, r_n$ uniformly at random from $\{0, \ldots, 2^{m+3} - 1\}$ and k from $\{1, \ldots, 2^{2m}\}$, then it computes the value of $p(r_1, \ldots, r_n)$ mod 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 $\frac{2}{3}$. Finally, if all the evaluations returned 0 as output the input is accepted; otherwise, it is rejected.

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 [3] and described in detail in the extended version of this paper [1].

As the input circuit is evaluated modulo some $k \in \mathbb{Z}$, the algorithm works in time polynomial with respect to $|x|$. Therefore, as a consequence of Theorem 4 and Lemma 8, there is a Σ_1^b-formula $G(x, y)$ of $\mathcal{RL}$ that represents it. The extended version of this paper [1] also contains a lower-level description of this formula G.

The Underlying Language. We show that there is a predicate H of $\mathcal{RL}$ such that $H(x,\epsilon)$ holds if and only if x is the encoding of a circuit in PZT; otherwise, $H(x,0)$ 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 1 and terminate.

For reasonable encodings of polynomial circuits and expressions, h is elementary recursive and therefore there is a predicate H which characterizes it, and $\mathrm{I}\Delta_0 + \mathrm{Exp}$ proves the totality of h. Moreover, we have $h = \chi_{\mathrm{PZT}}$, as for every polynomial p with coefficients in $\mathbb{Z}$, $\mathbb{Z} \models \forall \vec{x}.p(\vec{x}) = 0$ iff all the monomials in the normal form of p have zero as coefficient.

Proving the Error Bound. We now show that the formula G is not-erratic and that it decides $\mathrm{Lang}(\langle G \rangle)$. With the notations G^* and t_G from the the proof of Lemma 16, this can be reduced to proving in $\mathrm{I}\Delta_0 + \mathrm{Exp}$ the following two claims:

$$\vdash \forall z.\,(|z| = t_G(x) \wedge G^*(x,0,z)) \to H(x,0), \tag{$\dagger$}$$

$$\vdash \forall x.\left|\left\{z \preceq 2^{t_G(x)} \;\middle|\; G^*(x,\epsilon,z) \to H(x,\epsilon)\right\}\right| \geq \frac{2}{3}\cdot 2^{|2^{t_G(x)}|}. \tag{$\ddagger$}$$

($\dagger$) states that whenever the randomized algorithm rejects an input, then so does the deterministic one, while ($\ddagger$), which is reminiscent of ($\star$), states that in at least $\frac{2}{3}$ of all possible cases, if the randomized algorithm accepts the circuit, the deterministic one accepts it too. Jointly, ($\dagger$) and ($\ddagger$) imply that the equivalence $G^*(x,y,z) \leftrightarrow H(x,y)$ holds in at least $2/3$ of all possible cases.

While Claim ($\dagger$) is a consequence of the compatibility of the mod k function with addition and multiplication, which are easily proved in $\mathrm{I}\Delta_0 + \mathrm{Exp}$, the proof of Claim ($\ddagger$) is more articulated and relies on 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 (whose provability in $\mathrm{I}\Delta_0 + \mathrm{Exp}$ is known [16]) which bounds the probability to choose a *bad* value for k, i.e. one of those values causing `PZT` to return the wrong value. Detailed arguments are provided in the extended version of this paper [1].

Closure under Polytime Reduction. Only assessing that a problem belongs to $\mathbf{BPP}_{\mathrm{T}}$ does not tell us anything about other languages of this class; for this reason, we are interested in showing that $\mathbf{BPP}_{\mathrm{T}}$ is closed under polytime reduction. This allows us to start from $\mathrm{PZT} \in \mathbf{BPP}_{(\mathrm{I}\Delta_0+\mathrm{Exp})}$ to conclude that all problems which can be reduced to PZT in polynomial time belong to this class, and in particular that $\mathrm{PIT} \in \mathbf{BPP}_{(\mathrm{I}\Delta_0+\mathrm{Exp})}$. This is assessed by the following proposition, proved in the the extended version of this paper:

▶ **Proposition 25.** *For any theory* $\mathrm{T} \supseteq R\Sigma^b_1\text{-}NIA + \mathrm{Exp}$*, language* $L \in \mathbf{BPP}_{\mathrm{T}}$ *and language* $M \subseteq \mathbb{S}$*, if there is a polytime reduction from* M *to* L*, then* $M \in \mathbf{BPP}_{\mathrm{T}}$*.*

▶ **Corollary 26.** *PIT is in* $\mathbf{BPP}_{(\mathrm{I}\Delta_0+\mathrm{Exp})}$*.*

7 On Jeřábek's Characterization of BPP

As mentioned in Section 1, a semantic characterization of **BPP** based on bounded arithmetic was already provided by Jeřábek in [41]. This approach relies on checking, against the standard model, the truth of a formula which, rather than expressing that some machine is non-erratic, expresses what can be seen as a second totality condition (beyond the formula expressing the totality of the algorithm). Hence, also within this approach we think it makes sense to 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 $\mathbf{BPP}_{\mathrm{T}}$ are provably definable **BPP** problems, in the sense of [41], *within some suitable extension* of the bounded theory PV_1[13].

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 $\Pr_{w<r(x)}(A(x,w)) \leq p/q$. Jeřábek focuses on the theory PV_1, extended with an axiom schema dWPHP (PV_1)called the *dual weak pigeonhole principle* (cf. [41, pp. 962ff.]) for PV_1 (i.e. the axiom stating that for every PV_1-definable function f, f is not a surjection from x to x^2). The reason is that this theory is capable of proving *approximate* counting formulas of the form $\Pr_{w<r(x)}(A(x,w)) \preceq_0 p/q$, where "$\preceq_0$" is a relation equivalent to "$\leq$" up to some polynomially small error (recall that, in order to establish *exact* counting results, we were forced to use non-polytime operations, cf. Remark 17). 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 $S^1_2(PV)$ [13], which extends both PV_1 and Buss' S^1_2. Moreover, by arguing as in the proof of Lemma 14, in our characterization of **BPP** we can w.l.o.g. suppose that the formula G satisfies $\mathrm{EpsZero}[G] := \forall x.\forall y.G(x,y) \to y = \epsilon \vee y = 0$. Under this assumption, the de-randomization procedure described in the proof of Lemma 16 turns G into a pair (A, r), where $A = G^*$ is `Flip`-free and $r(x) = t_G(x)$, and the languages $L^+_{A,r}(x)$ and $L^-_{A,r}(x)$ correspond then to the formulas $L^+_G(x) := \mathsf{TwoThirds}[G(-,\epsilon)](x)$, and $L^-_G(x) := \mathsf{TwoThirds}[G(-,0)](x)$.

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

▶ **Proposition 27.** *Let L be a language with $L = \mathrm{Lang}(\langle G \rangle)$. If $L \in \mathbf{BPP}_{\mathrm{T}}$, then $\forall x.L^+_G(x) \vee L^-_G(x)$ is provable in some recursively enumerable extension of PV_1. Conversely, if $PV_1 + dWPHP(PV_1) \vdash \forall x.L^+_G(x) \vee L^-_G(x)$, then $L \in \mathbf{BPP}_{R\Sigma^b_1\text{-}NIA+\mathrm{Exp}}$.*

The second statement above relies on the fact that approximate counting can be replaced by exact counting in $R\Sigma^b_1$-NIA + Exp (i.e. "$\preceq_0$" can be replaced by "$\leq$").

8 Future Work

The authors see this work as a starting point for a long-term study on the logical nature of semantic classes. From this point of view, many ideas for further work naturally arise.

An exciting direction is the study of the expressiveness of the new syntactic classes $\mathbf{BPP}_{\mathrm{T}}$, 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, PV and PA, but also in theories which are *more expressive* than PA (like e.g. second-order theories). Surely, classes of the form $\mathbf{BPP}_{\mathrm{T}}$ could be analyzed also as for the existence of complete problems and hierarchy theorems for them, since such results are not known to hold for **BPP** itself [27].

Our approach to **BPP** suggests that extensions to other complexity classes of randomized algorithms like **ZPP**, **RP** and **coRP** could make sense. Notice that this requires to deal not only with error-bounds, but also with either average class complexity or with failure in decision procedures.

Finally, given the tight connections between bounded arithmetics and proof complexity, another natural direction is the study of applications of our work to randomized variations on the theme, for example recent investigations on *random resolution refutations* [41, 6, 52], i.e. resolution systems where proofs may make errors but are correct most of the time.

9 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 **BPP** and **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. In particular, the (difficult) challenge of enumerating **BPP** translates into the challenge of proving $\mathbf{BPP} = \mathbf{BPP}_{\mathrm{T}}$ for some strong enough r.e. theory T. So, it is worth exploring how much can be proved within expressive arithmetical theories. For this reason we focused here on a well-known problem, PIT, which is known to be in **BPP**, but not in **P**, showing that the whole argument for PIT $\in$ **BPP** can be formalized in a fragment of PA, namely $\mathrm{I}\Delta_0 + \mathrm{Exp}$.

References

1 M. Antonelli, U. Dal Lago, D. Davoli, I. Oitavem, and P. Pistone. Enumerating error bounded polytime algorithms through arithmetical theories, 2023. `arXiv:2311.15003`.

2 M. Antonelli, U. Dal Lago, and P. Pistone. On measure quantifiers in first-order arithmetic. In *Proc. of CiE 2021*, pages 12–24. Springer-Verlag, 2021.

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

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

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

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

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

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

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

10 A. Cobham. The intrinsic computational difficulty of functions. In *Proc. of the 1964 International Congress on Logics, Methodology and Philosophy of Science*, pages 24–30. North-Holland Publishing, 1965.

11 E.F. Codd. Relational Completeness of Data Base Sublanguages. In *Proc. of 6th Courant Computer Science Symposium.*, pages 65–98, 1972.

12 S. Cook. The Complexity of Theorem Proving Procedures. In *Proc. of STOC 1971*, pages 151–158, 1971.

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

14 S.A. Cook. Feasibly constructive proofs and the propositional calculus. In ACM Press, editor, *Proc. of STOC 1975*, pages 83–97, 1975.

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

16 C. Cornaros and C. Dimitracopoulos. The Prime Number Theorem and Fragments of PA. *Archive for Mathematical Logic*, 33:265–281, August 1994.

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

18 U. Dal Lago, R. Kahle, and I. Oitavem. A Recursion-Theoretic Characterization of the probabilistic Class PP. In *Proc. of MFCS 2021*, pages 1–12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

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

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

21 K. Eickmeyer and M. Grohe. Randomisation and Derandomisation in Descriptive Complexity Theory. In *Proc. of CSL 2010*. Springer, 2010.

22 R. Fagin. Generalized first-order spectra and polynomial-time recognizable sets. *Complexity of computation*, 7:43–73, 1974.

23 F. Ferreira. Polynomial-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 F. Ferreira. *Stockmeyer induction*, pages 161–180. Birkhäuser Boston, Boston, MA, 1990. doi:10.1007/978-1-4612-3466-1_9.

26 G. Ferreira and I. Oitavem. An Interpretation of S_2^1 in Σ_1^b-NIA. *Portugaliae Mathematica*, 63:137–156, 2006.

27 L. Fortnow. Comparing notions of full derandomization. In *Proceedings of the 16th Annual IEEE Conference on Computational Complexity*, pages 28–34, Chicago, IL, USA, 2001. IEEE Computer Society.

28 H. Gaifman and C. Dimitracopoulos. Fragments of Peano's arithmetic and the MRDP theorem. *Logic and Algorithmic, Monograph. Enseign. Math.*, 30:187–206, 1982.

29 D. Gajser. Verifying Time Complexity of Turing Machines. *Informatica*, 40:369–370, 2016.

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

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

32 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.

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

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

35 P. Hájek and P. Pudlák. *Metamathematics of First-Order Arithmetic*. Springer, Berlin-Heidelberg, 1998.

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

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

38 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.
39 N. Immerman. *Descriptive Complexity*. Springer, 1999.
40 E. Jeřábek. Dual Weak Pigeonhole Principle, Boolean Complexity, and Derandomization. *Annals of Pure and Applied Logic*, 129(1):1–37, 2004.
41 E. Jeřábek. Approximate Counting in Bounded Arithmetic. *Journal of Symbolic Logic*, 72(3):959–993, 2007.
42 J. Krajíček and P. Pudlák. Propositional Proof Systems, the Consistency of First-Order Theories and the Complexity of Computations. *Journal of Symboic Logic*, 54(3):1063–1079, 1989.
43 J. Krajíček, P. Pudlák, and G. Takeuti. Bounded Arithmetic and the Polynomial Hierarchy. *Annals of Pure and Applied Logic*, 52:143–153, 1991.
44 Y. Lafont. Soft Linear Logic and Polynomial Time. *Theoretical Computer Science*, 1/2(318):163–180, 2004.
45 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.
46 H. Michalewski and M. Mio. Measure Quantifiers in Monadic Second Order Logic. In *Proc. of LFCS*, pages 267–282, Cham, 2016. Springer.
47 J. Mitchell, M. Mitchell, and A. Scedrov. A Linguistic Characterization of Bounded Oracle Computation and Probabilistic Polynomial Time. In *Proc. of FOCS 1998*, pages 725–733. IEEE Computer Society, 1998.
48 C. Morgenstern. The Measure Quantifier. *Journal of Symbolic Logic*, 44(1):103–108, 1979.
49 R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, Cambridge; NY, 1995.
50 C.H. Papadimitriou. *Computational Complexity*. Pearson Education, 1993.
51 R. Parikh. Existence and Feasibility in Arithmetic. *Journal of Symbolic Logic*, 36:494–508, 1971.
52 P. Pudlák and N. Thapen. Random Resolution Refutations. *Computational Complexity*, 28:185–239, 2019.
53 E.S. Santos. Probabilistic Turing Machines and Computability. *AMS*, 22(3):704–710, 1969.
54 M.H. Sørensen and P. Urzyczyn. *Lectures on the Curry-Howard Isomorphism*. Elsevier, 2006.
55 A. Turing. On Computable Numbers, with an Application to the Entscheidungsproblem. *Proc. London Mathematical Society*, pages 2–42, 230–265, 1936-37.
56 G. Winskel. *The Formal Semantics of Programming Languages: An Introduction*. MIT press, 1993.