On Bounded Arithmetic and Randomized Complexity Classes

Anonymous author
Anonymous affiliation

Abstract

We introduce a minimal extension of the language of arithmetic, such that the bounded formulas provably total in a suitably-defined theory *à la Buss* precisely capture polynomial time random functions. Then, we use this language to provide new characterizations of the semantic class **BPP**, obtained by internalizing the error-bound check within the logic. We do that in two ways: one relies on measure quantifiers, the other encodes such quantifiers in a purely arithmetical language. This last result leads to introduce a family of effectively enumerable subclasses of **BPP** in which all provably non-entropic problems can be captured.

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

Computer science has been involved in numerous and profound interactions with mathematical logic since its early days (think of the seminal works of Turing [43] 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 [13, 30, 42]), the theory of databases (e.g. through Codd theorem [10]) and computational complexity (e.g. through descriptive complexity [3, 31]). In particular, this last discipline has to do with the so-called complexity classes [28, 9, 2], the nature of which still remains today, more than fifty years after the introduction of **P** and **NP**, essentially mysterious.

The possibility of describing fundamental complexity classes within the language of mathematical logic certainly offered a better understanding of the nature of such classes: since the seventies [19, 12], but especially from the eighties and nineties [6, 25, 3, 31, 33], 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 on explicit resource bounds. Moreover, characterizing a class by way of a simple enough proof-or-recursion theoretical system means being able to *enumerate* the problems in the class, and thus to devise a sound and complete language for the class, from which type systems and static analysis methodologies can be derived [29].

Among the various classes of problems with which computational complexity has been concerned, those defined on the basis of *randomized* algorithms [38] have so far proved more difficult to capture with the tools of logic. These include important and well-studied classes like **BPP** or **ZPP**. The former, for instance, 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 generally called a *semantic* class: for a (randomized) algorithm to be in **BPP**, it is not enough to be efficient, but one also has to check that the algorithm is not too *entropic*, that is once an input is fixed, it must produce each of the two possible output values with probabilities that are not too similar to each other. By their very nature, semantic classes like **BPP** are thus more challenging to capture through a logical system, compared to other (syntactic) classes like **P** or **NP**. Indeed, the sparse contributions along these lines are either themselves *semantic* [18], i.e. do not capture the limitations on the probability of error within the logical system [18, 16] (an interesting exception being [32]), or deal with classes, such as **PP**, which are not classifiable as semantic [14, 15].

In this paper we make a step towards a proper 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 Ferreira's $\mathcal{L}_{\mathbb{W}}$ [21] and in which formulas access a source of randomness through a distinguished unary predicate `Flip`, this way naturally capturing randomized algorithms. We even define a theory RS^1_2, which is the randomized analogue of Buss' theory S^1_2 [6] and Ferreira's Σ^b_1-NIA [21], and show that the functions which can be proved total in RS^1_2 are precisely the polytime *random* functions, i.e. those functions from strings to *distributions* of strings which can be computed by polytime probabilistic Turing machines. Using this result, we provide two characterizations of the algorithms in **BPP** by describing two different ways of controlling entropy: first, by means of *measure quantifiers* [37, 35, 1], i.e. well-studied second-order quantifiers capable of measuring *the extent* to which a formula is true; second, by showing that, when applied to bounded formulas, such quantifications can be reduced to standard first-order quantifications.

While both these approaches lead to precise characterizations of **BPP**, these are still of semantic nature: the entropy check is translated into conditions which are not checked within a formal system, but within the standard first-order model of arithmetics. Yet, we believe that the real novelty of our approach lies in the avenues it suggests. Notably, our arithmetization of **BPP** naturally leads to the introduction of a family of new *syntactic* subclasses $\mathbf{BPP}_T \subseteq \mathbf{BPP}$, made of languages for which the error-bounding condition is *provable* in a theory T. While it is unlikely that for some r.e. theory T, $\mathbf{BPP}_T = \mathbf{BPP}$, we conjecture that for some sufficiently expressive T, the class $\mathbf{BPP}_T$ should include problems, like polynomial identity testing, which are in **BPP** but not known to be in **P**.

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

- We introduce the arithmetical theory RS^1_2 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. To do so, we go through the definition of a class $\mathcal{POR}$ of *oracle recursive* functions, which is proved equivalent to the class of Σ^b_1-representable functions of RS^1_2 and, then, to the class **RFP** of probabilistic polynomial time functions. Technically, this is the most substantial result of this paper and is described in Section 4.
- We then prove that the aforementioned result leads to define two logical (but still semantic) characterizations of **BPP**. This is in Section 5.
- We conclude by briefly discussing the family of syntactic subclasses $\mathbf{BPP}_T \subseteq \mathbf{BPP}$, the proper investigation of which is left for future work. This is also in Section 5.

Related Work. As we have said, while the characterization of complexity classes through logic, and especially bounded arithmetic, is a vast research domain, there is not much related to probabilistic complexity classes. While several recursion-theoretic characterizations of the syntactic class **PP** exist [14, 15], concerning **BPP**, existing characterizations rely on some external, semantic condition [16, 36]. Eickmeyer and Grohe [18] provide another semantic characterization of **BPP** in a logic with fixed-point operators and a special counting quantifier, coupled with a probabilistic semantics not too different from the quantitative semantics we present in Section 3. By contrast, [32] studies a syntactic approach to probabilistic polytime programs in the context of bounded arithmetics and introduces a notion of "definable **BPP**-problem", relative to some bounded theory, and based on an arithmetical encoding of approximate counting problems. An intriguing question is whether such syntactically definable **BPP**-problems can be related to the syntactic classes $\mathbf{BPP}_T$ we define in Section 5.

2 Semantic Classes and their Characterization

Before delving into technical details, it is worth spending a few more words on the dichotomy between syntactic and semantic classes, and on the intrinsic difficulty of characterizing the latter. Although the literature does not offer a precise definition, this distinction appears in many popular textbooks (e.g., [2, 39]): on the one hand, semantic classes are those defined by imposing limitations on the amount of resources the underlying algorithm is allowed to use *but also* referring to a promise, typically that the underlying algorithm returns the correct answer *often enough*, on the other, in syntactic classes, the second condition is not present. In particular, in semantic classes being resource bounded is not enough for an algorithm to solve *some* problem in the class, since there can well be algorithms erring too often.

Among the semantic classes, we can certainly mention **BPP** and **ZPP**, while among the (many) syntactic ones, **P**, **NP**, and **PSPACE** are certainly among the most well-known. Notice that the dichotomy 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. By the way, since it could well be that $\mathbf{P} = \mathbf{BPP}$, the distinction cannot be an extensional one.

But now, why are syntactic classes relatively simple to characterize? This is due to the fact that, while it is very difficult to verify resource bounds on *arbitrary* algorithms, it is surprisingly easy to define an enumeration of resource bounded algorithms containing at least *one* algorithm for any problem in the class. To clarify what we mean, suppose we want to characterize a class like **P**. On the one hand, the class of *all* algorithms working in polynomial time is recursion-theoretically very hard, actually Σ^2_0-complete. On the other hand, the class of those algorithms 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 easy to enumerate and big enough to characterize **P**, at least in an extensional sense: every problem in **P** is decided by at least one algorithm in the class and vice versa. Many characterizations of **P** (and of other syntactic classes), as those based on safe-recursion [3], light and soft linear logic [24, 23, 34], and bounded arithmetic [6], can be seen as instances of the above pattern, where the precise class of polytime algorithms varies, leaving the underlying class of problems unchanged.

In semantic classes, that is, in presence of promises about the error, the enumeration strategy just sketched does not seem to be possible. How can we isolate a simple enough subclass of algorithms which are not only resource bounded, but also not too erratic? We believe that this paper makes a step forward in understanding the nature of this problem, without giving a definite answer. We show that bounded arithmetic can be adapted to randomized computation, a result which, although expected, is not so easy to prove, as explained in Section 4. But we go beyond that, showing how reasoning about error bounds can be internalized into the logic, giving rise to both precise, although semantic, characterizations of **BPP**, and a family of syntactic, but not necessarily precise, characterizations of the same class. The latter is, we believe, novel in its way of letting the whole power of an arithmetic theory T be used to capture the entropy of randomized computable functions.

3 From Arithmetic to Randomized Computation, Subrecursively

In this section we introduce the two main ingredients of our characterization of polytime randomized functions: a randomized bounded arithmetical theory RS^1_2 and a Cobham-style function algebra $\mathcal{POR}$ for polytime oracle recursive functions.

3.1 Recursive Functions and Arithmetical Formulas

Since the 1980s it is well-known that *bounded arithmetical theories*, i.e. subsystems of Peano Arithmetics where only *bounded quantifications* are admitted, can be used to characterize several complexity classes [6, 7]. At the core of these characterizations lies the fundamental result (known since Gödel's [26]) that recursive functions can be *represented* in Peano Arithmetics by means of Σ^0_1-formulas (i.e. formulas of the form $\exists x_1 . \dots . \exists x_n . A$, with A quantifier-free). For example, the following formula

$$A(x_1, x_2, y) := \exists x_3 . x_1 \times x_2 = x_3 \wedge y = \overline{s}(x_3)$$

represents the function $f(x_1, x_2) = (x_1 \times x_2) + 1$. Indeed, one can prove in PA that $\forall x_1 . \forall x_2 . \exists! y . A(x_1, x_2, y)$, i.e. , that A expresses a *functional* relation, and one can 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 $\mathbb{N}$) 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.

Our goal is to show that this approach can be extended also to classes of randomized computable functions. Our strategy focuses on a simple correspondence between *first-order predicates* over the natural numbers and *oracles* from the Cantor space $\{0,1\}^{\mathbb{N}}$. Indeed, suppose the aforementioned recursive function f has now the ability to observe a sequence ω of bits sampled from $\{0,1\}^{\mathbb{N}}$. For instance, f might observe the first bit of ω and return $(x_1 \times x_2) + 1$ if this bit is 0, and return 0 otherwise. Our idea is that we can capture the call by f to the oracle ω by adding, to the standard language of PA, a new unary predicate `Flip` (whose interpretation corresponds indeed to a stream of bits $\omega \in \{0,1\}^{\mathbb{N}}$). Our function f can then be represented by the following formula:

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

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 `Flip`, the actual numerical function that B represents depends on the choice of a value ω to interpret `Flip`, i.e. on the choice of an oracle for f.

The rest of this section is devoted to presenting the main ingredients of this correspondence, that will be made precise in Section 4.

The Language $\mathcal{RL}$. In the following we let $\mathbb{B} := \{\mathbf{0}, \mathbf{1}\}$ and let $\mathbb{S} := \mathbb{B}^*$ indicate the set of finite words from $\mathbb{B}$. Moreover, we let $\mathbb{O} := \mathbb{B}^{\mathbb{S}}$. Our first goal is to introduce a language for first-order arithmetics incorporating the new predicate symbol $\mathtt{Flip}(x)$ and its interpretation in the standard model. Following [22], we consider a first-order signature for natural numbers *in binary notation*. Consequently, formulas will be interpreted over $\mathbb{S}$ rather than $\mathbb{N}$.

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

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

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

The function symbol $\frown$ stands for string concatenation, while $t \times u$ indicates the concatenation of t with itself for as many times as the length of u. The binary predicate $\subseteq$ stands for the substring relation. As usual, we let $A \to B := \neg A \vee B$.

For readability we will use a series of 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, and $t|_r = s$ for $(\mathtt{1}^r \subseteq \mathtt{1}^t \wedge s \subseteq t \wedge \mathtt{1}^r = \mathtt{1}^s) \vee (\mathtt{1}^t \subseteq \mathtt{1}^r \wedge s = t)$, i.e. for the fact 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 (defined by $\overline{\epsilon} = \epsilon$, $\overline{\sigma\mathbf{0}} = \overline{\sigma}\mathtt{0}$ and $\overline{\sigma\mathbf{1}} = \overline{\sigma}\mathtt{1}$).
A defining feature of bounded arithmetic is the focus on so-called *bounded quantifications*. In $\mathcal{RL}$, *bounded quantifications*, abbreviated $\forall x \preceq t.A$, $\exists x \preceq t.A$, are of the form $\forall x.\mathtt{1}^x \subseteq \mathtt{1}^t \to F$ and $\exists x.\mathtt{1}^x \subseteq \mathtt{1}^t \wedge F$. Following [20], we call *subword quantifications* the quantifications of the form $\forall x \subseteq^* t.F$, $\exists x \subseteq^* t.F$, abbreviating $\forall x.(\exists w \subseteq t.wx \subseteq t) \to F$ and $\exists x.\exists w \subseteq t.wx \subseteq t \wedge F$. A formula F of $\mathcal{RL}$ 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 quantification is important 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 of $\mathcal{RL}$-formulas, 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 a formula A of $\mathcal{RL}$ the set $[\![A]\!] \subseteq \mathbb{O}$ of *all* possible interpretations of $\mathtt{Flip}$ that lead to satisfy A. Importantly, such sets $[\![A]\!]$ are *measurable*, a fact that will turn out essential in Section 5. 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 $b \in \mathbb{B}$, and the measure μ is uniquely defined by the condition $\mu(\mathsf{C}^b_\sigma) = \frac{1}{2}$ (see [4] and the Appendix).

▶ **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 of sequences $[\![F]\!]_\xi \in \sigma(\mathcal{C})$ inductively defined as follows:

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

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

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

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

Notice that 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.

Notice that an interpretation of the language $\mathcal{RL}$, in the usual first-order sense, is given by some ξ, in the sense of the definition 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, the sequence ω is in $[\![F]\!]_\xi$ precisely when $\xi + \omega \vDash F$ in the first-order sense.

The Bounded Theory RS^1_2. We now introduce a bounded theory RS^1_2 of the language $\mathcal{RL}$, which can be seen as a probabilistic counterpart to Ferreira's theory Σ^b_1-NIA [20]. The theory RS^1_2 is defined by axioms belonging to two classes:

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

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

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

- *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 of this schema to Σ^b_1-formulas, as in Buss' and Ferreira's approaches, is essential to characterize algorithms with bounded resources. Indeed, more general instances of this schema would lead to represent functions which are not polynomial time computable.

3.2 An Algebra of Polytime Oracle-Recursive Functions

We now introduce a Cobham-style function algebra for polytime *oracle* recursive functions. This algebra is inspired from Ferreira's algebra $\mathcal{PTCA}$ [20, 21]. 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 $\omega : \mathbb{S} \longrightarrow \mathbb{B}$, to be interpreted as the underlying source of random bits, and include the basic function *query*, $Q(x, \omega) = \omega(x)$, which can be used to observe any bit from ω.

More precisely, the *class* $\mathcal{POR}$ is the smallest class of functions from $\mathbb{S}^n \times \mathbb{O}$ to $\mathbb{S}$, containing the *empty* function $E(x, \omega) = \boldsymbol{\epsilon}$, the *projection* functions $P^n_i(x_1, \dots, x_n, \omega) = x_i$, the *word-successor* $S_{\mathbf{b}}(x, \omega) = x\mathbf{b}$, for every $\mathbf{b} \in \mathbb{B}$, the *conditional* function defined by $C(\boldsymbol{\epsilon}, y, z_{\mathbf{0}}, z_{\mathbf{1}}, \omega) = y$ and $C(x\mathbf{b}, y, z_{\mathbf{0}}, z_{\mathbf{1}}, \omega) = z_{\mathbf{b}}$, where $\mathbf{b} \in \mathbb{B}$, the *query* function $Q(x, \omega) = \omega(x)$, and closed under the following schemata:

- *Composition*, where f is defined from $g, h_1, \dots, h_k$ as $f(\vec{x}, \omega) = g(h_1(\vec{x}, \omega), \dots, h_k(\vec{x}, \omega), \omega)$.
- *Bounded recursion on notation*, where f is defined from g, h_0, h_1 as

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

  where t is obtained from $\boldsymbol{\epsilon}, \mathbf{0}, \mathbf{1}, \frown, \times$ by explicit definition, that is, t can is obtained applying $\frown$ and $\times$ on the constants $\boldsymbol{\epsilon}$, $\mathbf{0}$, $\mathbf{1}$, and the variables $\vec{x}$ and y (notice that this language is identical to the one of $\mathcal{RL}$ terms, cf. Def. 1).

4 Characterizing Polytime Random Functions

In this section, we outline our main result, namely that the bounded theory RS^1_2 provides a characterization of polytime random functions. Usual characterizations of polytime (deterministic) functions in bounded arithmetic are obtained in two steps [6, 11, 20]. First, some Cobham-style algebra for polytime functions is introduced and its functions are shown to be precisely those which are Σ^b_1-*representable*. Then, it is proved that functions in the Cobham-style algebra correspond to those computed by TMs running in polynomial time.

Our proof follows a similar path, with the algebra $\mathcal{POR}$ playing the role of our Cobham-style functional algebra. We start by showing that the random functions which are Σ^b_1-representable in RS^1_2 are precisely those in $\mathcal{POR}$. Then, we establish that $\mathcal{POR}$ is equivalent (in a very specific sense) to the class of functions computed by PTMs running in polynomial time. While the first part of the argument closely follows a standard argument for the deterministic case [20, 11], the presence of randomness introduced a novel and delicate ingredient to be considered in the second part. Indeed, functions in $\mathcal{POR}$ access randomness in a rather different way with respect to PTMs and relating these different probabilistic computational models requires some effort, involving the construction of long chains of intermediate simulations (illustrated in Fig. 1).

4.1 RS^1_2 characterizes $\mathcal{POR}$

The first step consists in showing that $\mathcal{POR}$ functions are precisely those which are Σ^b_1-representable in RS^1_2. To do so, we need to extend Buss' representability condition by adding a constraint that links the quantitative semantics of formulas in RS^1_2 with the additional functional parameter ω of the $\mathcal{POR}$ functions.

▶ **Definition 1.** *A function* $f : \mathbb{S}^k \times \mathbb{O} \to \mathbb{S}$ *is* Σ^b_1-representable in RS^1_2 *if there exists a* Σ^b_1*-formula* $G(\vec{x}, y)$ *of* $\mathcal{RL}$ *such that:*

1. $RS^1_2 \vdash \forall \vec{x}.\exists! y.G(\vec{x}, y)$ *,*
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* $\omega \in [\![G(\overline{\vec{\sigma}}, \overline{\tau})]\!]$.

Then, the first part of our argument is expressed by the statement below:

▶ **Theorem 2.** *For any function $f : \mathbb{S}^k \times \mathbb{O} \to \mathbb{S}$, f is Σ^b_1-representable in RS^1_2 iff $f \in \mathcal{POR}$.*

Proof sketch. One direction of the argument constructs, by induction on the structure of the $\mathcal{POR}$ function algebra, a Σ^b_1-formula representing the desired function. Notice that the formula $\forall \vec{x}.\exists! y.G(\vec{x}, y)$ occurring in condition 1. of Def. 1 is *not* Σ^b_1, since its 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 [40] (which holds for S^1_2) to RS^1_2, i.e. if $RS^1_2 \vdash \forall \vec{x}.\exists y.G(\vec{x}, y)$, where $G(\vec{x}, y)$ is a Σ^b_1-formula, then we can find a term t such that $RS^1_2 \vdash \exists y \preceq t.G(\vec{x}, y)$.

The converse direction, detailed in the Appendix, is proved by adapting the proof by Cook and Urquhart for the system IPV^ω [11], and passes through a *realizability interpretation* of the intuitionistic version of RS^1_2, called IRS^1_2. ◀

4.2 $\mathcal{POR}$ and Polytime Probabilistic Turing Machines

Theorem 2 shows that it is possible to characterize a class of polytime random functions by means of a system of bounded arithmetic. However, this result is not enough as our final goal is to characterize probabilistic classes, like **BPP** or **RP**, which are defined in terms of functions computed by a PTM. As we observed before, there is a crucial difference in the way in which PTMs and $\mathcal{POR}$ functions access randomness, so our next goal is to fill this gap, by relating such classes of functions in a precise way.

Let us first define the class of functions computed by polytime PTMs.

▶ **Definition 3** (Class **RFP**). *Let $\mathbb{D}(\mathbb{S})$ indicate the set of* distributions on $\mathbb{S}$, *that is, those functions $\lambda : \mathbb{S} \to [0, 1]$ such that $\sum_{\sigma \in \mathbb{S}} \lambda(\sigma) = 1$. The class* **RFP** *is made of all functions $f : \mathbb{S}^k \to \mathbb{D}(\mathbb{S})$ such that, for some PTM ν running in polynomial time, and every $\sigma_1, \dots, \sigma_k, \tau \in \mathbb{S}$, $f(\sigma_1, \dots, \sigma_k)(\tau)$ coincides with the probability that $\nu(\sigma_1 \sharp \dots \sharp \sigma_k) \Downarrow \tau$.*

In other words, the function computed by a TM associates each possible output with a probability corresponding to the actual probability that a run of the machine will actually produce that output. While such functions have a different shape from those considered so far, it is still possible to define a notion of Σ^b_1-representability for them, relying on the fact that any closed formula A of RS^1_2 generates a *measurable* set $[\![A]\!] \subseteq \mathbb{B}^{\mathbb{N}}$.

▶ **Definition 4.** *A function $f : \mathbb{S}^k \to \mathbb{D}(\mathbb{S})$ is* Σ^b_1-representable in RS^1_2 *if there exists a Σ^b_1-formula $G(\vec{x}, y)$ of $\mathcal{RL}$ such that:*
1. *$RS^1_2 \vdash \forall \vec{x}.\exists! y.G(\vec{x}, y)$,*
2. *for all $\sigma_1, \dots, \sigma_k, \tau \in \mathbb{S}$, $f(\sigma_1, \dots, \sigma_k, \tau) = \mu\big([\![G(\overline{\vec{\sigma}}, \overline{\tau})]\!]\big)$.*

The final result of this section can now be stated as follows:

▶ **Theorem 5.** *For any function $f : \mathbb{S}^k \to \mathbb{D}(\mathbb{S})$, f is Σ^b_1-representable in RS^1_2 iff $f \in$* **RFP**.

Actually, Theorem 5 can be deduced from the corresponding result for $\mathcal{POR}$ (i.e. Theorem 2) once we can relate the functional algebra $\mathcal{POR}$ with the class **RFP**:

▶ **Lemma 6.** *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.*

The proof of the Lemma 6, which concludes our argument, is convoluted. The rest of this section provides an overview of this argument, which rests on a chain of language simulations.

The first fundamental idea to relate $\mathcal{POR}$ and **RFP** is to introduce an intermediate class **SFP**, corresponding to the set of functions computed by polytime *Stream Turing Machines* (STM for short). Am STM is defined as a deterministic TM with one extra read-only tape which, intuitively, accounts for probabilistic choices: at the beginning of the computation the extra tape is sampled from $\mathbb{B}^{\mathbb{N}}$, and then at each computation step the machine reads one new bit from this tape, always moving to the right.

► **Definition 7** (Class **SFP**). *The class* **SFP** *is composed of those functions* $f : \mathbb{S}^k \times \mathbb{B}^{\mathbb{N}} \to \mathbb{S}$ *such that for some STM* ν *running in polynomial time and for all* $\sigma_1, \dots, \sigma_k$ *and* $\eta \in \mathbb{B}^{\mathbb{N}}$, $f(\sigma_1, \dots, \sigma_k, \eta) = \tau$ *iff the machine* ν*, with inputs* $\sigma_1 \sharp \dots \sharp \sigma_k$ *and tape* η*, outputs* τ*.*

STMs behave similarly to PTMs, but access to randomness is now *explicit*: instead of flipping a coin at each step, the machine samples a stream of bits once, and then reads one new bit at each step. The equivalence of the two models is expressed by the result below.

► **Proposition 8** (Equivalence of PTMs and STMs). *For any polytime STM* ν *there exists a polytime PTM* ν^* *such that for all string* $\sigma, \tau \in \mathbb{S}$, $\mu(\{\eta \mid \nu(\sigma, \eta) = \tau\}) = \Pr[\nu^*(\sigma) = \tau]$*, and conversely.*

The following result immediately follows from Proposition 8:

► **Corollary 9** (Equivalence of **RFP** and **SFP**). *For any function* $f : \mathbb{S}^k \to \mathbb{D}(\mathbb{S})$ *in* **RFP** *there is a function* $g : \mathbb{S}^k \times \mathbb{B}^{\mathbb{N}} \to \mathbb{S}$ *in* **SFP** *such that for all* $\sigma_1, \dots, \sigma_k, \tau \in \mathbb{S}$, $f(\sigma_1, \dots, \sigma_k, \tau) = \mu(\{\eta \mid g(\sigma_1, \dots, \sigma_k, \eta) = \tau\})$*, and conversely.*

It thus remains to show a form of equivalence between the classes $\mathcal{POR}$ and **SFP**.

From SFP to $\mathcal{POR}$. With the replacement of PTMs by STMs, we now have a notion of probabilistic machine which accesses randomness in a similar way to $\mathcal{POR}$-functions: at the beginning of the computatio,n an oracle is sampled, and the computation then proceeds by possibly querying this oracle. Yet, there are still important differences in the way in which these computation models treat randomness. First, while $\mathcal{POR}$-functions access an oracle in the form of a function $\omega \in \mathbb{O} = \mathbb{B}^{\mathbb{S}}$, the oracle for an STM is a stream of bits $\eta \in \mathbb{B}^{\mathbb{N}}$. In other words, while a $\mathcal{POR}$-function is of the form $f : \mathbb{S}^k \times \mathbb{B}^{\mathbb{S}} \to \mathbb{S}$, an **SFP**-function is of the form $f : \mathbb{S}^k \times \mathbb{B}^{\mathbb{N}} \to \mathbb{S}$, and thus we cannot compare them *directly*. Instead, we provide an indirect comparison of the form below:

► **Proposition 10** (From **SFP** to $\mathcal{POR}$). *For any* $f : \mathbb{S}^k \times \mathbb{B}^{\mathbb{N}} \to \mathbb{S}$ *in* **SFP** *there exists a function* $f^\sharp : \mathbb{S}^k \times \mathbb{O} \to \mathbb{S}$ *such that for all* $\sigma_1, \dots, \sigma_k, \tau \in \mathbb{S}$,

$$\mu\big(\{\eta \in \mathbb{B}^{\mathbb{N}} \mid f(\sigma_1, \dots, \sigma_k, \eta) = \tau\}\big) = \mu\big(\{\omega \in \mathbb{O} \mid f^\sharp(\sigma_1, \dots, \sigma_k, \omega) = \tau\}\big).$$

The function $f^\sharp$ is constructed in several steps. The fundamental observation is that, given an input $n \in \mathbb{S}$ and extra-tape $\eta \in \mathbb{B}^{\mathbb{N}}$, an STM running in polynomial time can access a *finite* portion of η only, the length of which can be bounded by some polynomial $p(|n|)$. Using this fact, we construct $f^\sharp$ as follows:

1. We introduce a new class of functions **PTF** of the form $f : \mathbb{S}^k \times \mathbb{S} \to \mathbb{S}$, which are computed by a variant of STM, called *Finite Stream Turing Machines* (FSTMs), defined like STMs, but with the extra-tape now being *finite*, i.e. corresponding to a finite string.
2. Given a function $f : \mathbb{S} \times \mathbb{B}^{\mathbb{N}} \to \mathbb{S}$ with polynomial bound $p(x)$, we define a function $h : \mathbb{S} \times \mathbb{S} \to \mathbb{S}$ in **PTF** such that $f(n, \eta) = h(x, \eta_{p(|x|)})$.

3. We now define a function $h' : \mathbb{S} \times \mathbb{S} \times \mathbb{O} \to \mathbb{S}$ such that $h'(x, y, \omega) = h(x, y)$ and show, by an encoding of FSTMs, that $h' \in \mathcal{POR}$ (and moreover, h' can be defined *without* using the query function, since the computation of h' never actually looks at ω).
4. Finally, we define an *extractor function* $e : \mathbb{S} \times \mathbb{O} \to \mathbb{S}$ in $\mathcal{POR}$, which mimics the prefix extractor $\eta_{p(|x|)}$, in the sense that its outputs have *the same distributions* of all possible η's prefixes, even though within a different space (recall that $\eta \in \mathbb{B}^{\mathbb{N}}$, while the second argument of e is in $\mathbb{O}$). This is obtained by exploiting a bijection $dyad : \mathbb{S} \to \mathbb{N}$, ensuring that for each $\eta \in \mathbb{B}^{\mathbb{N}}$ there is an $\omega \in \mathbb{B}^{\mathbb{S}}$ such that any prefix of η is an output of $e(y, \omega)$ for some y. Since $\mathcal{POR}$ is closed under composition, we can finally define $f^{\sharp}(x, \omega) := h'(x, e(x, \omega), \omega)$.

From $\mathcal{POR}$ to SFP. Our goal is now to establish a converse of Proposition 10. In order to simulate $\mathcal{POR}$-functions via STMs we must consider that not only these two models invoke oracles of different shape, but also that the former can manipulate such oracles in a much more liberal way than the latter:

- an STM must query its oracle before each step is produced; by contrast $\mathcal{POR}$-functions may invoke the query function $Q(x, \omega)$ freely during computation. We call this an *on-demand* access policy.
- At each step of computation an STM queries a new bit of the oracle, and cannot access previously observed bits; we call this a *linear* access policy; by contrasts, $\mathcal{POR}$-functions can query the same bits as many times as needed.

It is for these reasons that a direct simulation of $\mathcal{POR}$ via STMs looks challenging, even for a basic function like the $Q(x, \omega)$. So, to define it we exploit again an indirect path, and pass through a chain of simulations, dealing with each of these differences separately. We summarize this chain of simulations below (see also Fig. 1):

1. The first step translates $\mathcal{POR}$ into an imperative language $\mathbf{SIFP_{RA}}$ inspired from Winskell's IMP [44] and with the *same* access policy as $\mathcal{POR}$: $\mathbf{SIFP_{RA}}$ is endowed with assignments, a `while` construct, and a command $\texttt{Flip}(e)$, which first evaluates e to a string n and then stores the value $\omega(n)$ in a register. The encoding of $\mathcal{POR}$-functions in $\mathbf{SIFP_{RA}}$ is easily obtained by induction on the function algebra.
2. Then, we translate $\mathbf{SIFP_{RA}}$ into another imperative language $\mathbf{SIFP_{LA}}$ with *linear* access policy: $\mathbf{SIFP_{LA}}$ is defined like $\mathbf{SIFP_{RA}}$, except for the command $\texttt{Flip}(e)$, replaced by a new command $\texttt{RandBit}()$ which generates a random bit and stores it in a register. A weak simulation from $\mathbf{SIFP_{RA}}$ into $\mathbf{SIFP_{LA}}$ is defined by progressively constructing an *associative table* containing pairs $(\text{string}, \text{bit})$ of past observations: each time $\texttt{Flip}(e)$ is invoked, the simulation checks if a pair (e, b) had already been observed, and otherwise increments the table by producing a new pair $(e, \texttt{RandBit}())$. This is by far the most complex step of the whole simulation.
3. The language $\mathbf{SIFP_{LA}}$ can now be translated into STMs. However, notice that the access policy of $\mathbf{SIFP_{LA}}$ is still on-demand: $\texttt{RandBit}()$ may be invoked or not before executing an instruction. To obviate with this obstacle, we first consider a translation from $\mathbf{SIFP_{LA}}$ into a variant of STMs admitting an on-demand access policy (that is, a computation step may or may not access a bit from the extra-tape). Then, the resulting program is encoded into a regular STM. Observe that also in this last, apparently trivial case, we cannot expect that the machine $\nu^{\dagger}$ simulating an on-demand machine ν will produce *the same* output given the same input and the same oracle (i.e. $\nu^{\dagger}(\sigma, \omega) = \nu(\sigma, \omega)$); rather, as in many other cases above, we show that $\nu^{\dagger}$ can be defined so that for all strings σ, τ, the sets $\{\omega \mid \nu^{\dagger}(\sigma, \omega) = \tau\}$ and $\{\omega \mid \nu(\sigma, \omega) = \tau\}$ have the same measure.

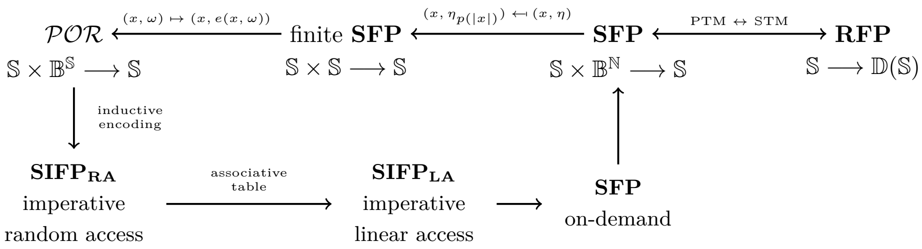

Figure 1 Structure of the proof of equivalence between $\mathcal{POR}$ and **RFP** via **SFP**.

5 Towards BPP

In this section, we turn our attention to the class **BPP**. By relying on the correspondence between RS^1_2 and **RFP** we provide two semantic characterizations of this class: the first one relies on the use of *measure quantifiers* [1], while the second one is purely arithmetical and rests on the possibility of arithmetizing such quantifiers. Finally, by relying on the latter, we introduce a family of syntactic classes $\mathbf{BPP}_T \subseteq \mathbf{BPP}$, containing those languages whose belonging to **BPP** is provable in a sufficiently expressive theory T.

5.1 BPP via Measure Quantifiers

As discussed in Sections 1 and 2, in order to check if a language belongs to the class **BPP** one has to look for a probabilistic algorithm satisfying *both* a polynomial resource bound and a uniform error bound. Indeed, let us recall the definition of this class:

▶ **Definition 3 (BPP).** A language $L \subseteq \mathbb{S}$ is in **BPP** if and only if, said f_L the characteristic function of L, there is a Polynomial Probabilistic Turing Machine M such that $\forall \sigma \in L.Pr[M(\sigma) = f_L(\sigma)] \geq \frac{2}{3}$.

Given the correspondence established in the previous section, a natural question is whether the theory RS^1_2, or some extension of it, can be used also to measure error bounds for probabilistic algorithms. We will show that this is indeed the case, but, once more, we will do this in a series of progressive steps.

Given that any formula F of $\mathcal{RL}$ is associated with a measurable set $[\![F]\!] \subseteq \mathbb{O}$, a first natural idea is to enrich this language with *measure quantifiers* $\mathbf{C}^q.F$ [1], where $q \in [0,1] \cap \mathbb{Q}$, with the intuitive meaning that $\mathbf{C}^q.F$ holds whenever $[\![F]\!]$ has measure at least q. Such quantifiers have been studied in mathematical logic since [37]; more recently, they have been applied to the study of probabilistic programs [35, 1].

Let $\mathcal{RL}^{\mathsf{MQ}}$ indicate the extension of the language $\mathcal{RL}$ with quantifiers of the form $\mathbf{C}^{\frac{t}{s}}.F$, where t, s are terms. The Borel semantics of $\mathcal{RL}$ extends naturally to $\mathcal{RL}^{\mathsf{MQ}}$ by letting

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

For readability, for all $n, m \in \mathbb{N}$, we let $\mathbf{C}^{\frac{n}{m}}.F$ abbreviate $\mathbf{C}^{\frac{1^n}{1^m}}.F$.

To characterize **BPP**, first observe that, since $\mathbf{BPP} \subseteq \mathsf{PH}$, for any language $L \in \mathbf{BPP}$, the corresponding characteristic function $f_L : \mathbb{S} \to \mathbb{B}$ is represented by some formula $H_L(x, y)$ of PA. Actually, using results from Buss and Goldreich [6, 7, 27] (and suitably adapting them to our framework) the formula $H_L(x,y)$ can be taken to be a Σ^b_3 formula of $\mathcal{RL}$.[1] This leads to the following characterization:

▶ **Theorem 11** (First Semantic Characterization of $\mathbf{BPP}$). *For any language $L \subseteq \mathbb{S}$, $L \in \mathbf{BPP}$ iff there exists a Σ^b_1-formula $G(x,y)$ such that the following hold:*
1. $RS^1_2 \vdash \forall x.\exists! y.G(x,y)$.
2. $\forall \sigma \in \mathbb{S}, b \in \mathbb{B}, \vDash \mathbf{C}^{\frac{2}{3}}.G(\overline{\sigma},\overline{b}) \leftrightarrow H_L(\overline{\sigma},\overline{b})$.

Proof Sketch. Suppose $L \in \mathbf{BPP}$ and let $g : \mathbb{S} \to \mathbb{D}(\mathbb{S})$ be a $\mathbf{RFP}$-function computing L with a uniform error bound. Using Theorem 5, there is a Σ^b_1-formula $G(x,y)$ such that for all σ, b, $\mu(\llbracket G(\overline{\sigma},\overline{b})\rrbracket) \geq 2/3$ iff $g(\sigma)(b) \geq 2/3$. Then, for all σ, if $f_L(\sigma)=0$, we deduce $\llbracket H_L(\overline{\sigma},\mathbf{0})\rrbracket = \mathbb{O}$ and $g(\sigma)(b) \geq 2/3$, and thus $\mu(\llbracket G(\overline{\sigma},\mathbf{0})\rrbracket) \geq 2/3$, so we conclude $\mu(\llbracket G(\overline{\sigma},\mathbf{0}) \leftrightarrow H_L(\overline{\sigma},\mathbf{0})\rrbracket) \geq 2/3$. If $f_L(\sigma)=1$, we can argue in a similar way. Conversely, if (1) and (2) hold, then there is a function $g \in \mathbf{RFP}$ such that for all σ, b, $g(\sigma)(b) = \mu(\llbracket G(\overline{\sigma},\overline{b})\rrbracket)$. If $f_L(\sigma)=0$ then $\llbracket H_L(\overline{\sigma},\mathbf{0})\rrbracket = \mathbb{O}$ and thus by (2) $\llbracket G(\overline{\sigma},\mathbf{0})\rrbracket \geq 2/3$, whence by (1) $g(\sigma)(0) \geq 2/3$, and similarly if $L(\sigma)=1$. We conclude then that $L \in \mathbf{BPP}$. ◀

5.2 Arithmetizing Measure Quantifiers

Theorem 11 relies on a tight correspondence between first-order arithmetic and probabilistic computation; yet, the fundamental condition (2) exploits a class of formulas which are based on an essentially measure-theoretical operator, and thus are not arithmetical formulas.

However, the following lemma shows that measure quantifications over bounded formulas of $\mathcal{RL}$ can be expressed in an arithmetical language. Let $\mathcal{RL}^{\exp}$ be the language obtained from $\mathcal{RL}$ by (1) adding a new unary symbol $\mathbf{2}^x$ and (2) eliminating the predicate $\texttt{Flip}(x)$. The interpretation of $\mathbf{2}^x$ is the function $h : \mathbb{S} \to \mathbb{S}$ given by $h(\epsilon) = \mathbf{1}$ and $h(\sigma b) = h(\sigma)h(\sigma)$.

▶ **Lemma 12.** *For any bounded formula $F(\vec{x})$ of $\mathcal{RL}$ there exists a Σ^b_3-formula $\mathsf{TwoThirds}[F](\vec{x})$ of $\mathcal{RL}^{\exp}$ such that for all $\vec{\sigma} \in \mathbb{S}$, $\vDash \mathsf{TwoThirds}[F](\overline{\vec{\sigma}})$ holds iff $\mu(\llbracket F(\overline{\vec{\sigma}})\rrbracket) \geq \frac{2}{3}$.*

Proof Sketch. The construction of the formula $\mathsf{TwoThirds}[F](\vec{x},y)$ is made in several steps. First, we observe that, for any bounded formula $F(\vec{x})$ of $\mathcal{RL}$ and $\omega \in \mathbb{O}$, the portion of bits of ω that one has to observe in order to check whether $\omega \in \llbracket F\rrbracket$ is finite. More precisely, one can construct a term $t(\vec{x})$ of $\mathcal{RL}^{\exp}$ such that for all $\vec{\sigma} \in \mathbb{S}$ and $\omega, \omega' \in \mathbb{O}$, if $\omega|_{|t(\vec{\sigma})|} = \omega'|_{|t(\vec{\sigma})|}$, then $\omega \in \llbracket F(\sigma)\rrbracket$ iff $\omega' \in \llbracket F(\sigma)\rrbracket$. Then, using the fact above, measuring $\llbracket F(\sigma)\rrbracket$ is reduced to *counting* the elements of a finite set of strings of length $|t(\vec{\sigma})|$. This set can be captured by transforming the formula $F(\vec{x})$ into a formula $\mathsf{NoFlip}[F](\vec{x},y)$ of $\mathcal{RL}^{\exp}$ (hence eliminating all occurrences of $\texttt{Flip}(x)$) intuitively expressing that y is a string of length $|t(\vec{x})|$ encoding the initial segment of some ω satisfying $F(\vec{x})$. Using so-called *threshold quantifiers* $\exists^{\geq t}x.F$ ("there are *at least* t distinct x such that F") the original formula $F(\vec{x})$ is thus converted to $\exists^{\geq u_{2/3}}y.\mathsf{NoFlip}[F](\vec{x},y)$, where $u_{2/3}$ is a large enough term as to correspond to the 2/3 probability requirement.

Finally, it is shown that threshold quantification $\exists^{\geq w}y.F(\vec{x},y)$ over a Σ^b_1-formula can be encoded via a Σ^b_3-formula $\exists y \preceq u_F.\mathsf{Threshold}[F](\vec{x},y,w)$, so we can define $\mathsf{TwoThirds}[F](\vec{x}) :=$ $\exists y \preceq u_{\mathsf{NoFlip}[F]}.\mathsf{Threshold}[\mathsf{NoFlip}[F](\vec{x},y)](\vec{x},y,u_{2/3})$. ◀

Now, Theorem 11 and Lemma 12 yield a purely arithmetical characterization of $\mathbf{BPP}$:

[1] That is, a formula of the form $F(\vec{x}) = \exists y \preceq t(\vec{x}).\forall z \preceq u(\vec{x},y).F'(\vec{x},y,z)$, with $F'(\vec{x},y,z)$ a Σ^b_1-formula.

▶ **Theorem 13** (Second Semantic Characterization of $\mathbf{BPP}$). *For any language* $L \subseteq \mathbb{S}$, $L \in \mathbf{BPP}$ *iff there exists a* Σ_1^b*-formula* $G(x, y)$ *such that the following hold:*

1. $RS_2^1 \vdash \forall x.\exists! y.G(x, y)$.
2. $\models \forall x.\forall y.\mathsf{TwoThirds}[G(x, y) \leftrightarrow H_L(x, y)]$.

5.3 Provable BPP Problems

The characterization provided by Theorem 13 is still semantical, since the crucial condition (2) is not checked within a formal system, but over the standard model of RS_2^1. Yet, as soon as the entropy condition is expressed in a purely arithmetical language, it makes sense to consider *syntactic* variants of Condition (2), where the check on $\mathbb{S}$ is replaced by some sufficiently expressive theory.

▶ **Definition 14** (Class $\mathbf{BPP}_T$). *Let* $T \supseteq RS_2^1$ *be a theory in the language* $\mathcal{RL} \cup \mathcal{RL}^{\exp}$. *The* class $\mathbf{BPP}$ relative to T *contains all languages* $L \subseteq \mathbb{S}$ *such that for some* Σ_1^b*-formula* $G(x, y)$ *the following hold:*

1. $RS_2^1 \vdash \forall x.\exists! y.G(x, y)$.
2. $T \vdash \forall x.\forall y.\mathsf{TwoThirds}[G(x, y) \leftrightarrow H_L(x, y)]$.

Whenever T is sound (i.e. $T \vdash A$ implies that A is true in the standard model), it is clear that $\mathbf{BPP}_T \subseteq \mathbf{BPP}$. Conversely, we do not know if one can find some recursively enumerable theory T such that also $\mathbf{BPP}_T \supseteq \mathbf{BPP}$ holds, but this seems unlikely. Indeed, a crucial difference between the syntactic class $\mathbf{BPP}_T$ and $\mathbf{BPP}$ is that, when T is r.e., the algorithms of $\mathbf{BPP}_T$ can be *enumerated* (by enumerating the proofs of (1) and (2) in T), while, as we discussed before, it does not seem possible to define an enumeration of randomized algorithms containing a witness for any problem in $\mathbf{BPP}$. Yet, we conjecture that some interesting r.e. subclasses $\mathbf{BPP}_T$ may contain important problems in $\mathbf{BPP}$ which are not know to be in $\mathbf{P}$.

6 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 some convenient tools to control error bounds, another essential ingredient in the definition of classes like $\mathbf{BPP}$ and $\mathbf{ZPP}$.

Among the many questions that this work leaves open, an exciting direction is the study of the expressiveness of the new syntactic classes $\mathbf{BPP}_T$. When T is sufficiently expressive, $\mathbf{BPP}_T$ should include well-known $\mathbf{BPP}$ problems, like polynomial identity testing, for which no exact polytime algorithm is known. Similarly, it would be interesting to determine for which theories T the classes $\mathbf{BPP}_T$ and $\mathbf{BPP}$ coincide.

Given the tight connections between bounded arithmetics and proof complexity, another natural direction is to study applications of our work to probabilistic approaches in this field, for example on recent work on *random resolution refutations* [32, 5, 41], 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. Del Lago, and P. Pistone. On Measure Quantifiers in First-Order Arithmetic. In L. De Mol, A. Weiermannn, 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. *Comput. 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 Yehoshua Bar-Hillel, editor, *Logic, Methodology and Philosophy of Science: Proceedings 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 and A. Urquhart. Functional Interpretations of Feasibly Constructive Arithmetic. *Annals of Pure and Applied Logic*, 63(2):103–200, 1993.
12. S.A. Cook and R.A. Reckhow. Efficiency of Propositional Proof Systems. *The J. of Symbolic Logic*, 44(1):36–50, 1979.
13. H. B. Curry. Functionality in Combinatory Logic*. *Proc. of the National Academy of Sciences*, 20(11):584–590, 1934.
14. U. Dal Lago, R. Kahle, and I. Oitavem. A Recursion-Theoretic Characterization of the probabilistic Class PP. In *Proc. of MFCS*, pages 1–12, 2021.
15. U. Dal Lago, R. Kahle, and I. Oitavem. Implicit Recursion-Theoretic Characterizations of Counting Classes. *Archive for Mathematical Logic*, May 2022.
16. U. Dal Lago and P. Parisen Toldin. A higher-order characterization of probabilistic polynomial time. *Information and Computation*, 241:114–141, 2015.
17. D. Davoli. Bounded Arithmetic and Randomized Computation. Master's thesis, University of Bologna, 2022. URL: `http://amslaurea.unibo.it/26234/`.
18. 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.
19. R. Fagin. Generalized First-Order Spectra and Polynomial-Time Recognizable Sets. In *Complexity of Computing: SIAM-AMS Proc.*, pages 43–73, 1974.
20. F. Ferreira. Polyonmial-Time Computable Arithmetic and Conservative Extesions. Ph.D. Dissertation, December 1988.
21. F. Ferreira. Polynomial-Time Computable Arithmetic. In W. Sieg, editor, *Logic and Computation*, volume 106 of *Contemporary Mathematics.* AMS, 1990.
22. G. Ferreira and I. Oitavem. An Interpretation of S^1_2 in Σ^b_1-NIA. *Portugaliae Mathematica*, 63:427–450, 2006.
23. J.-Y. Girard. Light Linear Logic. *Information and Computation*, 2(143):175–204, 1998.
24. J.-Y. Girard and Y. Lafont. *Advances in Linear Logic.* Cambridge University Press, 1995.
25. J.-Y. Girard, A. Scedrov, and P. Scott. Bounded Linear Logic: A Modular Approach to Polynomial-Time Computability. *TCS*, 97(1):1–66, 1992Textbook.

26 K. Gödel. Über Formal Unentscheidbare Sätze der Principia Mathematica and Verwandter Systeme. *Monatshefte für Mathematik und Physik*, 38:173–198, 1931.
27 O. Goldreich and D. Zuckerman. Another proof that BPP $\subseteq$ PH (and more). *Electronic Colloquium on Computational Complexity - ECCC*, 01 1997. doi:10.1007/978-3-642-22670-0_6.
28 J. Hartmanis and R.E. Stearns. On the Computational Complexity of Algorithms. *Transactions of the AMS*, 117:285–306, 1965.
29 M. Hofmann. Programming Languages Capturing Complexity Classes. *SIGACT News*, 31(1):31–42, mar 2000.
30 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.
31 N. Immerman. *Descriptive Complexity.* Springer, 1999.
32 E. Jeràbek. Approximate Counting in Bounded Arithmetic. *The J. of Symbolic Logic*, 72(3):959–993, 2007.
33 J. Krajicek and P. Pudlak. Propositional Proof Systems, the Consistency of First-Order Theories and the Complexity of Computations. *J. of Symboic Logic*, 54(3):1063–1079, 1989.
34 Y. Lafont. Soft Linear Logic adn Polynomial Time. *Theoretical Computer Science*, 1/2(318):163–180, 2004.
35 H. Michalewski and M. Mio. Measure Quantifiers in Monadic Second Order Logic. In *Proc. of LFCS 2016*, pages 267–282, Cham, 2016. Springer.
36 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.
37 C. Morgenstern. The Measure Quantifier. *J. Symb. Log.*, 44(1):103–108, 1979.
38 R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, Cambridge; NY, 1995.
39 C.H. Papadimitriou. *Computational Complexity.* Pearson Education, 1993.
40 R. Parikh. Existence and feasibility in arithmetic. *Journal of Symbolic Logic*, 36:494–508, 1971.
41 P. Pudlak and N. Thapen. Random resolution refutations. *Computational Complexity*, 28:185–239, 2019.
42 M.H. Sorensen and P. Urzyczyn. *Lectures on the Curry-Howard Isomorphism.* Elsevier, 2006.
43 A. Turing. On Computable Numbers, with an Application to the Entscheidungsproblem. *Proc. London Mathematical Society*, pages 2–42, 230–265, 1936-37.
44 G. Winskel. *The Formal Semantics of Programming Languages: an Introduction.* MIT press, 1993.

A Proofs from Section 4.1

A.1 Proof of Theorem 2($\Leftarrow$)

In order to prove Theorem 2($\Leftarrow$), we introduce a slightly-modified version of Parikh's theorem [40], which is usually presented in the context of Buss' bounded arithmetics, as stating that given a Boolean formula F (in $\mathcal{L}_\mathbb{N}$) such that $S_2^i \vdash \forall \vec{x}.\exists y.B(\vec{x}, y)$, then there is a term $t(\vec{x})$ such that $S_2^i \vdash \forall \vec{x}.\exists y \preceq t(\vec{x}).B(\vec{x}, y)$ [7].

▶ **Proposition 1.** Let $F(\vec{x}, y)$ be a bounded formula in $\mathcal{RL}$ such that $RS_2^1 \vdash (\forall \vec{x})(\exists y)F(\vec{x}, y)$. Then, there is a term t such that $RS_2^1 \vdash \forall \vec{x}.\exists y \preceq t(\vec{x}).F(\vec{x}, y)$.

Proof of Theorem 2($\Leftarrow$). The proof is by induction on the structure of functions in $\mathcal{POR}$. We consider the query function only as all other cases are standard. $f = Q$ is Σ_1^b-representable in RS_2^1 by the formula

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

Notice that the proof relies on the fact that every $f \in \mathcal{POR}$ invokes *exactly one* oracle.

1. Existence is proved by cases. Intuitively if $RS^1_2 \vdash \texttt{Flip}(x)$, we let $y = \texttt{1}$. By the reflexivity of identity $RS^1_2 \vdash \texttt{1} = \texttt{1}$ holds, so $RS^1_2 \vdash \texttt{Flip}(x) \wedge \texttt{1} = \texttt{1}$. Then, we conclude by purely-logical rules $RS^1_2 \vdash \exists y.(\texttt{Flip}(x) \wedge \texttt{1} = \texttt{1}) \vee (\neg\texttt{Flip}(x) \wedge \texttt{1} = \texttt{0})$. If $RS^1_2 \vdash \neg\texttt{Flip}(x)$, we let $y = \texttt{0}$ and proceed int he same way.
2. Uniqueness is established relying on the transitivity of identity.
3. For every $n, m \in \mathbb{S}$ and $\omega^* \in \mathbb{O}$, $Q(n, \omega^*) = m$ iff $\omega^* \in [\![G_Q(\overline{n}, \overline{m})]\!]$. Assume $m = \mathbf{1}$. Then $Q(n, \omega^*) = \mathbf{1}$, i.e. $\omega^*(n) = \mathbf{1}$,

$$\begin{aligned}
[\![(\texttt{Flip}(\overline{n}) \wedge \overline{m} = \texttt{1}) \vee (\neg\texttt{Flip}(\overline{n}) \wedge \overline{m} = \texttt{0})]\!] &= [\![\texttt{Flip}(\overline{n}) \wedge \overline{m} = \texttt{1}]\!] \cup [\![\neg\texttt{Flip}(\overline{n}) \wedge \overline{m} = \texttt{0}]\!] \\
&= ([\![\texttt{Flip}(\overline{n})]\!] \cap [\![\texttt{1} = \texttt{1}]\!]) \cup ([\![\neg\texttt{Flip}(\overline{n})]\!] \cap [\![\texttt{1} = \texttt{0}]\!]) \\
&= ([\![\texttt{Flip}(\overline{n})]\!] \cap [\![\texttt{1} = \texttt{1}]\!]) \cup ([\![\neg\texttt{Flip}(\overline{n})]\!] \cap \emptyset) \\
&= [\![\texttt{Flip}(n)]\!] \\
&= \{\omega \mid \omega(n) = \mathbf{1}\}.
\end{aligned}$$

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

Inductive cases are also standard. For bounded recursion the proof is especially convoluted, following [20]. ◀

A.2 Proof of Theorem 2($\Rightarrow$)

The proof of Theorem 2($\Rightarrow$) adapts the strategy used by Cook and Urquhart for IPV$^\omega$ [11] and is structured as follows:

1. We introduce a basic equational theory $\mathcal{POR}^\lambda$ for a simply typed λ-calculus with primitives corresponding to functions of $\mathcal{POR}$.
2. We define the *intuitionistic* theory $\mathcal{IPOR}^\lambda$, extending $\mathcal{POR}^\lambda$ with usual predicate calculus and an **NP**-induction schema, and IRS^1_2, which is the intuitionistic version of RS^1_2. We show $\mathcal{IPOR}^\lambda$ able to prove all theorems of IRS^1_2.
3. We develop a realizability interpretation of $\mathcal{IPOR}^\lambda$ (inside itself): for any derivation of $\forall x.\exists y.A(x, y)$, with $A \in \Sigma^b_0$, a λ-term $\mathtt{t}$ of $\mathcal{POR}^\lambda$ can be extract, such that $\vdash_{\mathcal{IPOR}^\lambda} \forall x.A(x, \mathtt{t})$. We show that every function which is Σ^b_1-representable in IRS^1_2 is in $\mathcal{POR}$.
4. Finally, we extend this result to classical RS^1_2, showing that any Σ^b_1-formula provable in $\mathcal{IPOR}^\lambda$ + Excluded Middle (EM, for short) is already provable in $\mathcal{IPOR}^\lambda$.

The Theory $\mathcal{POR}^\lambda$

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

Types of $\mathcal{POR}^\lambda$ are standard, while terms are obtained by adding to simply typed λ-terms constants from the signature below:

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

Intuitively, $\mathsf{Tail}(x)$ computes the string obtained by deleting the first digit of x, $\mathsf{Trunc}(x, y)$ that obtained by truncating x at the length of y, $\mathsf{Cond}(x, y, z, w)$ the function that yields y

if $x = \boldsymbol{\epsilon}$, z if $x = x'\mathbf{0}$ and w if $x = x'\mathbf{1}$, $\mathsf{Flipcoin}(x)$ indicates a random $\mathbf{0}/\mathbf{1}$ generator, and Rec is the operator for bounded recursion on notation.

▶ **Notation 1.** In the following, we often define terms implicitly using bounded recursion on notation. We now introduce a few abbreviations for composed functions:

- $\mathsf{B}(x) := \mathsf{Cond}(x, \epsilon, \mathsf{0}, \mathsf{1})$ indicates the function computing the last digit of x.
- $\mathsf{BNeg}(x) := \mathsf{Cond}(x, \epsilon, \mathsf{1}, \mathsf{0})$ computes the Boolean negation of $\mathsf{B}(x)$.
- $\mathsf{BOr}(x, y) := \mathsf{Cond}(\mathsf{B}(x), \mathsf{B}(y), \mathsf{B}(y), \mathsf{0})$ coerces x, y to Booleans and performs OR-operation.
- $\mathsf{BAnd}(x, y) := \mathsf{Cond}(\mathsf{B}(x), \epsilon, \mathsf{0}, \mathsf{B}(y))$ coerces x, y to Booleans and performs AND-operation.
- $\mathsf{Eps}(x) := \mathsf{Cond}(x, \mathsf{1}, \mathsf{0}, \mathsf{0})$ indicates the characteristic function of "$x = \epsilon$".
- $\mathsf{Bool}(x) := \mathsf{BAnd}(\mathsf{Eps}(\mathsf{Tail}(x)), \mathsf{BNeg}(\mathsf{Eps}(x)))$ is the characteristic function of "$x = \mathsf{0} \vee x = \mathsf{1}$".
- $\mathsf{Zero}(x) := \mathsf{Cond}(\mathsf{Bool}(x), \mathsf{0}, \mathsf{Cond}(x, \mathsf{0}, \mathsf{0}, \mathsf{1}), \mathsf{0})$ is the characteristic function of the predicate "$x = \mathsf{0}$".
- $\mathsf{Conc}(x, y)$ indicates the concatenation function.
- $\mathsf{Eq}(x, y)$ is the characteristic function of "$x = y$" and is defined by double recursion.
- $\mathsf{Times}(x, y)$ is the function for self-concatenation, $x, y \mapsto x \times y$.
- $\mathsf{Sub}(x, y)$ is the initial-substring function $x, y \mapsto S(x, y)$.

$\mathcal{POR}^\lambda$ is reminiscent of PV^ω [11] (without the induction rule R5), the main difference being the constant $\mathsf{Flipcoin}$, denoting a function which randomly generates either $\mathbf{0}$ or $\mathbf{1}$. Formulas of $\mathcal{POR}^\lambda$ are all equations $\mathsf{t} = \mathsf{u}$, where t, u are terms of type s.

▶ **Definition 4** (The Theory $\mathcal{POR}^\lambda$)**.** Axioms of $\mathcal{POR}^\lambda$ are the following ones:

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

$$\begin{aligned}
\epsilon x = x\epsilon &= x \\
x(y\mathsf{b}) &= (xy)\mathsf{b} \\
\\
\mathsf{Tail}(\epsilon) &= \epsilon \\
\mathsf{Tail}(x\mathsf{b}) &= x \\
\\
\mathsf{Trunc}(x, \epsilon) = \mathsf{Trunc}(\epsilon, x) &= \epsilon \\
\mathsf{Trunc}(x\mathsf{b}, y\mathsf{0}) = \mathsf{Trunc}(x\mathsf{b}, y\mathsf{1}) &= \mathsf{Trunc}(x, y)\mathsf{b} \\
\\
\mathsf{Cond}(\epsilon, y, z, w) &= y \\
\mathsf{Cond}(x\mathsf{0}, y, z, w) &= z \\
\mathsf{Cond}(x\mathsf{1}, y, z, w) &= w \\
\\
\mathsf{Bool}(\mathsf{Flipcoin}(x)) &= \mathsf{1} \\
\\
\mathsf{Rec}(x, h_0, h_1, k, \epsilon) &= x \\
\mathsf{Rec}(x, h_0, h_1, k, y\mathsf{0}) &= \mathsf{Trunc}(h_0 y(\mathsf{Rec}(x, h_0, h_1, k, y)), ky) \\
\mathsf{Rec}(x, h_0, h_1, k, y\mathsf{1}) &= \mathsf{Trunc}(h_1 y(\mathsf{Rec}(x, h_0, h_1, k, y)), ky),
\end{aligned}$$

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

- The (β)- and (ν)-axioms:

$$\mathsf{C}\big[(\lambda x.\mathsf{t})\mathsf{u}\big] = \mathsf{C}\big[\mathsf{t}\{\mathsf{u}/x\}\big] \tag{β}$$

$$\mathsf{C}\big[\lambda x.\mathsf{t}x\big] = \mathsf{C}[\mathsf{t}] \tag{ν}$$

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 $\mathsf{C}[\;]$.

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

$$\mathsf{t} = \mathsf{u} \vdash \mathsf{u} = \mathsf{t} \tag{R1}$$

$$\mathsf{t} = \mathsf{u}, \mathsf{u} = \mathsf{v} \vdash \mathsf{t} = \mathsf{v} \tag{R2}$$

$$\mathsf{t} = \mathsf{u} \vdash \mathsf{v}\{\mathsf{t}/x\} = \mathsf{v}\{\mathsf{u}/x\} \tag{R3}$$

$$\mathsf{t} = \mathsf{u} \vdash \mathsf{t}\{\mathsf{v}/x\} = \mathsf{u}\{\mathsf{v}/x\}. \tag{R4}$$

Let $\vdash_{\mathcal{POR}^\lambda} \mathsf{t} = \mathsf{u}$ indicate that the equation $\mathsf{t} = \mathsf{u}$ is deducible by instances of axioms and inference rules above. Given a set of equations T, $T \vdash_{\mathcal{POR}^\lambda} \mathsf{t} = \mathsf{u}$ indicates that $\mathsf{t} = \mathsf{u}$ is deducible using the given axioms and rules plus equations from T.

For any string $s \in \mathbb{S}$ and $\omega \in \mathbb{O}$, $\overline{\overline{s}} : s$ denotes the term of $\mathcal{POR}^\lambda$ corresponding to it, i.e. $\overline{\overline{\boldsymbol{\epsilon}}} = \epsilon$, $\overline{\overline{s\mathbf{0}}} = \overline{\overline{s}}\mathsf{0}$, $\overline{\overline{s\mathsf{1}}} = \overline{\overline{s}}\mathsf{1}$, and T_ω is the set of all equations of the form $\mathsf{Flipcoin}(\overline{\overline{s}}) = \overline{\overline{\omega(s)}}$.

▶ **Definition 5** (Provable Representability). Let $f : \mathbb{O} \times \mathbb{S}^k \to \mathbb{S}$. A term $\mathsf{t} : s \Rightarrow \ldots \Rightarrow s$ of $\mathcal{POR}^\lambda$ *provably represents* f when for all strings $s_1, \ldots, s_j, s \in \mathbb{S}$ and $\omega \in \mathbb{O}$,

$$f(s_1, \ldots, s_n, \omega) = s \Leftrightarrow T_\omega \vdash_{\mathcal{POR}^\lambda} \mathsf{t}\overline{\overline{s_1}} \ldots \overline{\overline{s_j}} = \overline{\overline{s}}.$$

▶ **Example 1.** The term $\mathsf{Flipcoin} : s \Rightarrow s$ provably represents the query function $Q(x, \omega) = \omega(x)$ of $\mathcal{POR}$. Indeed, for any $s \in \mathbb{S}$ and $\omega \in \mathbb{O}$, $\mathsf{Flipcoin}(\overline{\overline{s}}) = \overline{\overline{\omega(s)}} \vdash_{\mathcal{POR}^\lambda} \mathsf{Flipcoin}(\overline{\overline{s}}) = \overline{\overline{Q(s, \omega)}}$.

Notice that terms $\mathsf{Tail}, \mathsf{Trunc}, \mathsf{Cond}$ provably represents $f_{\mathsf{Tail}}, f_{\mathsf{Trunc}}$ and Cond, resp., where $f_{\mathsf{Tail}}(s, \omega)$ is the string obtained by chopping the first digit of s and $f_{\mathsf{Trunc}}(s_1, s_2, \omega) = s_1|_{s_2}$.

▶ **Theorem 15.** 1. *Any function $f \in \mathcal{POR}$ is provably represented by a term $\mathsf{t} \in \mathcal{POR}^\lambda$.*
2. *For any $\mathsf{t} \in \mathcal{POR}^\lambda$, there is an $f \in \mathcal{POR}$ such that f is provably represented by t.*

Proof Sketch. ($\Rightarrow$) The proof is by induction on the structure of $f \in \mathcal{POR}$.
($\Leftarrow$) As a consequence of the normalization, a β-normal term $\mathsf{t} : s \Rightarrow \ldots \Rightarrow s$ cannot contain variables of higher types and each possible normal form represents functions in $\mathcal{POR}$. ◀

▶ **Corollary 1.** For any function $f : \mathbb{S}^j \times \mathbb{O} \to \mathbb{S}$, $f \in \mathcal{POR}$ when f is provably represented by some $\mathsf{t} : s \Rightarrow \ldots \Rightarrow s \in \mathcal{POR}^\lambda$.

The Theory $\mathcal{IPOR}^\lambda$

The theory $\mathcal{POR}^\lambda$ is rather weak, as, for example, one cannot prove even simple equations as $x = \mathsf{Tail}(x)\mathsf{B}(x)$ (as some form of induction is needed). So, we introduce $\mathcal{IPOR}^\lambda$, which extends $\mathcal{POR}^\lambda$ with basic predicate calculus and a restricted induction principle. We also define the intuitionistic version of RS^1_2, the so-called IRS^1_2. We show that all theorems of $\mathcal{POR}^\lambda$ and IRS^1_2 are provable in $\mathcal{IPOR}^\lambda$ and the latter provides a language to associate derivations in IRS^1_2 with polytime computable functions, corresponding to $\mathcal{IPOR}^\lambda$-terms.

▶ **Definition 6** (Formulas of $\mathcal{IPOR}^\lambda$)**.** (i) All equations $\mathtt{t} = \mathtt{u}$ of $\mathcal{POR}^\lambda$ are formulas of $\mathcal{IPOR}^\lambda$, (ii) given (possibly open) term $\mathtt{t}, \mathtt{u} : s \in \mathcal{POR}^\lambda$, $\mathtt{t} \subseteq \mathtt{u}$ and $\mathtt{Flip}(\mathtt{t})$ are formulas of $\mathcal{IPOR}^\lambda$, (iii) formulas of $\mathcal{IPOR}^\lambda$ are closed under $\wedge, \vee, \rightarrow, \forall, \exists$.

▶ **Notation 2.** We define $\bot := \mathsf{0} = \mathsf{1}$ and $\neg A := A \rightarrow \bot$. Furthermore, any formula of RS^1_2 can be seen as a formula of $\mathcal{IPOR}^\lambda$ where each occurrence of $\mathsf{0}$ is replaced by $\mathtt{0}$, $\mathsf{1}$ by $\mathtt{1}$, $\frown$ by $\circ$, and $\times$ by Times. In the following, we suppose that any formula of RS^1_2 is a formula of $\mathcal{IPOR}^\lambda$.

▶ **Definition 7** (Theory $\mathcal{IPOR}^\lambda$)**.** Axioms and inference rules of $\mathcal{IPOR}^\lambda$ include the standard rule of the intuitionistic first-order predicate calculus, usual rules for equality and the axioms below: (1) all axioms of $\mathcal{POR}^\lambda$, (2) $x \subseteq y \leftrightarrow \mathsf{Sub}(x, y) = \mathtt{1}$, (3) $x = \epsilon \vee x = \mathsf{Tail}(x)\mathtt{0} \vee x = \mathsf{Tail}(x)\mathtt{1}$, (4) $\mathtt{0} = \mathtt{1} \rightarrow x = \epsilon$, (5) $\mathsf{Cond}(x, y, z, w) = w' \leftrightarrow (x = \epsilon \wedge w' = y) \vee (x = \mathsf{Tail}(x)\mathtt{0} \wedge w' = z) \vee (x = \mathsf{Tail}(x)\mathtt{1} \wedge w' = w)$, (6) $\mathtt{Flip}(x) \leftrightarrow \mathsf{Flipcoin}(x) = \mathtt{1}$, (7) Any formula of the form $(A(\epsilon) \wedge \forall x.(A(x) \rightarrow A(x\mathtt{0})) \wedge \forall x.(A(x) \rightarrow A(x\mathtt{1}))) \rightarrow \forall y.A(y)$, where A is of the form $\exists z \preceq \mathtt{t}.\mathtt{u} = \mathtt{v}$, with $\mathtt{t}$ containing only first-order open variables.

▶ **Notation 3** (**NP**-Predicate)**.** We will refer to a formula in the form $\exists z \preceq \mathtt{t}.\mathtt{u} = \mathtt{v}$, with $\mathtt{t}$ containing only first-order open variables, as an **NP**-predicate.

It is now possible to show that all theorems of both $\mathcal{POR}^\lambda$ and IRS^1_2 are derivable in $\mathcal{IPOR}^\lambda$. In particular, Prop 2 is proved by systematic inspection of $\mathcal{POR}^\lambda$-rules.

▶ **Proposition 2.** Any theorem of $\mathcal{POR}^\lambda$ is a theorem of $\mathcal{IPOR}^\lambda$.

To prove that every theorem of IRS^1_2 is derivable in $\mathcal{IPOR}^\lambda$ we need to establish a few useful properties concerning $\mathcal{IPOR}^\lambda$. In particular, the recursion schema of $\mathcal{IPOR}^\lambda$ differs from that of IRS^1_2 as dealing with formulas of the form $\exists y \preceq \mathtt{t}.\mathtt{u} = \mathtt{v}$ rather than all the Σ^b_1-ones.

▶ **Proposition 3.** For any Σ^b_0-formula $A(x_1, \ldots, x_n)$ of $\mathcal{RL}$, there is a term $\mathtt{t}_A(x_1, \ldots, x_n)$ of $\mathcal{POR}^\lambda$ such that: (i) $\vdash_{\mathcal{IPOR}^\lambda} A \leftrightarrow \mathtt{t}_A = \mathtt{0}$, (ii) $\vdash_{\mathcal{IPOR}^\lambda} \mathtt{t}_A = \mathtt{0} \vee \mathtt{t}_A = \mathtt{1}$.

▶ **Corollary 2.** i. For any Σ^b_0-formula A, $\vdash_{\mathcal{IPOR}^\lambda} A \vee \neg A$.
ii. For any closed Σ^b_0-formula A and $\omega \in \mathbb{O}$, either $T_\omega \vdash_{\mathcal{IPOR}^\lambda} A$ or $T_\omega \vdash_{\mathcal{IPOR}^\lambda} \neg A$.

So we conclude,

▶ **Theorem 16.** *Any theorem of IRS^1_2 is a theorem of $\mathcal{IPOR}^\lambda$.*

Proof. For any Σ^b_1-formula $A = \exists x_1 \preceq t_1 \ldots \exists x_n \preceq t_n.B$ of IRS^1_2, $\vdash_{\mathcal{IPOR}^\lambda} A \leftrightarrow \exists x_1 \preceq \mathtt{t}_1 \ldots \exists x_n \preceq \mathtt{t}_n.\mathtt{t}_B = \mathtt{0}$. So, any instance of the Σ^b_1-recursion schema of IRS^1_2 is derivable in $\mathcal{IPOR}^\lambda$ from the **NP**-inductions schema. To prove that $\mathcal{IPOR}^\lambda$ extends IRS^1_2, it suffices to check that all basic axioms of IRS^1_2 are provable in $\mathcal{IPOR}^\lambda$. ◀

Furthermore, due to Corollary 2, we establish Lemma 17 below.

▶ **Lemma 17.** *Given a closed Σ^b_0-formula A in $\mathcal{RL}$ and $\omega \in \mathbb{O}$, $T_\omega \vdash_{\mathcal{IPOR}^\lambda} A$ iff $\omega \in [\![A]\!]$.*

Realizability

We introduce realizability as internal to $\mathcal{IPOR}^\lambda$ to show that for any derivation in IRS^1_2 (actually, in $\mathcal{IPOR}^\lambda$) of a formula $\forall x.\exists y.A(x, y)$, we extract a functional term $\mathsf{f} : s \Rightarrow s$ of $\mathcal{POR}^\lambda$, such that $\vdash_{\mathcal{IPOR}^\lambda} \forall x.A(x, \mathsf{f}x)$. So, we conclude that if f is Σ^b_1-representable in IRS^1_2, then $f \in \mathcal{POR}$.

▶ **Notation 4.** Let $\mathbf{x}, \mathbf{y}$ denote finite sequences of term variables, $\mathbf{x}(\mathbf{y})$ 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 8.** Formulas $x \circledR A$ are defined by induction on the structure of A:

$$\begin{aligned}
\Lambda \circledR A &:= A \quad (A \text{ atomic}) \\
\mathbf{x}, \mathbf{y} \circledR (B \wedge C) &:= (\mathbf{x} \circledR B) \wedge (\mathbf{y} \circledR C) \\
z, \mathbf{x}, \mathbf{y} \circledR (B \vee C) &:= (z = 0 \wedge \mathbf{x} \circledR B) \vee (z \neq 0 \wedge \mathbf{y} \circledR C) \\
\mathbf{y} \circledR (B \to C) &:= \forall \mathbf{x}.((\mathbf{x} \circledR B) \to \mathbf{y}(\mathbf{x}) \circledR C) \wedge (B \to C) \\
z, \mathbf{x} \circledR \exists y.B &:= \mathbf{x} \circledR B\{z/y\} \\
\mathbf{x} \circledR \forall y.B &:= \forall y.(\mathbf{x}(y) \circledR B),
\end{aligned}$$

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

We can now link derivability of such formulas with that of formulas in $\mathcal{IPOR}^\lambda$.

▶ **Theorem 18** (Soundness and Completeness). *i If $\vdash_{\mathcal{IPOR}^\lambda} \mathbf{t} \circledR A$, then $\vdash_{\mathcal{IPOR}^\lambda} A$.*
ii. If $\vdash_{\mathcal{IPOR}^\lambda} A$, then there exist $\mathbf{t}$ such that $\vdash_{\mathcal{IPOR}^\lambda} \mathbf{t} \circledR A$.

Proof Sketch. Proofs are by induction on formulas ($i.$) and on derivation height ($ii.$). ◀

▶ **Corollary 3.** Let $\forall x.\exists y.A(x, y)$ be a closed theorem of $\mathcal{IPOR}^\lambda$, where A is a Σ^b_1-formula. Then, there is a closed term $\mathsf{t} : s \Rightarrow s$ of $\mathcal{POR}^\lambda$, such that: $\vdash_{\mathcal{IPOR}^\lambda} (\forall x)A(x, \mathsf{t}x)$

Proof. By Theorem 18.ii, there is $\mathbf{w} = \mathsf{t}, w$ such that $\vdash_{\mathcal{IPOR}^\lambda} \mathbf{w} \circledR (\forall x)(\exists y)A(x, y)$,

$$\begin{aligned}
\mathbf{w} \circledR (\forall x)(\exists y)A(x, y) &\equiv (\forall x)(\mathbf{w}(x) \circledR (\exists y)A(x, y)) \\
&\equiv (\forall x)(w(x) \circledR A(x, \mathsf{t}x)).
\end{aligned}$$

From this, by Theorem 18.i, we conclude $\vdash_{\mathcal{IPOR}^\lambda} \forall x.A(x, \mathsf{t}x)$. ◀

We can now prove that if a function is Σ^b_1-representable in IRS^1_2, then it is in $\mathcal{POR}$.

▶ **Corollary 4.** For any function $f : \mathbb{O} \times \mathbb{S} \to \mathbb{S}$, if there is a closed Σ^b_1-formula $A(x, y)$ in $\mathcal{RL}$ such that (1) $IRS^1_2 \vdash \forall x.\exists! y.A(x, y)$, (2) $[\![A(\overline{\overline{s_1}}, \overline{\overline{s_2}})]\!] = \{\omega \mid f(\omega, s_1) = s_2\}$, then $f \in \mathcal{POR}$.

Proof. Since $\vdash_{IRS^1_2} \forall x.\exists! y.A(x, y)$, by Theorem 16, also $\vdash_{\mathcal{IPOR}} \forall x.\exists! y.A(x, y)$, from which we deduce $\vdash_{\mathcal{IPOR}^\lambda} \forall x.A(x, \mathsf{g}x)$ for some closed term $\mathsf{g} : s \Rightarrow s$ of $\mathcal{POR}^\lambda$, by Cor 3 and by Theorem 15, there is a function $g \in \mathcal{POR}$ such that for any $\omega \in \mathbb{O}$ and $s_1, s_2 \in \mathbb{S}$, $T_\omega \vdash_{\mathcal{IPOR}^\lambda} A(\overline{\overline{s_1}}, \overline{\overline{s_2}}) \Leftrightarrow g(s_1, \omega) = s_2$. From this we conclude,

$$\begin{aligned}
g(s_1, \omega = s_2) \quad &\Leftrightarrow \quad T_\omega \vdash_{\mathcal{IPOR}^\lambda} A(\overline{\overline{s_1}}, \overline{\overline{s_2}}) \\
&\Leftrightarrow \quad \omega \in [\![A(\overline{\overline{s_1}}, \overline{\overline{s_2}})]\!] \\
&\Leftrightarrow \quad f(s_1, \omega) = s_2.
\end{aligned}$$

So, since $f = g$, we conclude that $f \in \mathcal{POR}$. ◀

Concluding the Proof

To conclude we need to extend Corollary 4 to classical RS^1_2, showing that any function which is Σ^b_1-representable in RS^1_2 is also in $\mathcal{POR}$. We start by generalizing $\mathcal{IPOR}^\lambda$ via EM, $A \vee \neg A$. We show that realizability interpretation extends to such $\mathcal{IPOR}^\lambda$+EM, so that for any of its closed theorems $\forall x.\exists y \preceq \mathsf{t}.A(x, y)$, with $A \in \Sigma^b_1$, there is a closed term $\mathsf{t} : s \Rightarrow s$ of $\mathcal{POR}^\lambda$, such that $\vdash_{\mathcal{IPOR}^\lambda} \forall x.A(x, \mathsf{t}x)$. To do so, we pass through Markov's principle.

▶ **Definition 9** (Markov's Principle). For any $A \in \Sigma^b_1$, Markov's principle is defined as:

$$\neg\neg\exists x.A \to \exists x.A. \qquad \text{(Markov)}$$

▶ **Proposition 4.** For any Σ^b_1-formula A, if $\vdash_{\mathcal{IPOR}^\lambda+EM} A$, then $\vdash_{\mathcal{IPOR}^\lambda+(Markov)} A$.

Proof Sketch. The proof relies on double-negation translation. Notice that for any Σ^b_0-formula A, $\vdash_{\mathcal{IPOR}^\lambda} \neg\neg A \to A$. ◀

Now, we need to show that the realizability interpretation extends to $\mathcal{IPOR}^\lambda$+(Markov), that is for any of its closed theorems $\forall x.\exists y \preceq \mathsf{t}.A(x,y)$, with $A \in \Sigma^b_1$, there is a closed term $\mathsf{t} : s \Rightarrow s$ of $\mathcal{POR}^\lambda$, such that $\vdash_{\mathcal{IPOR}^\lambda} \forall x.A(x,\mathsf{t}x)$. Then, given a subjective 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

$$x * y := \sharp(\lambda z.\mathsf{BAnd}(\mathsf{app}(x,z), \mathsf{app}(y,z)))$$
$$T(x) := \exists y.\mathsf{B}(\mathsf{app}(x,y)) = \mathsf{0}.$$

There is a *meet semi-lattice* structure on the set of terms of type s defined by $\mathsf{t} \sqsubseteq \mathsf{u}$ iff $\vdash_{\mathcal{IPOR}^\lambda} T(\mathsf{u}) \to T(\mathsf{t})$ 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 $\mathsf{B}(\mathsf{app}(x,\mathsf{u})) = \mathsf{0}$, we obtain $\mathsf{B}(\mathsf{app}(x{*}y,\mathsf{u})) = \mathsf{BAnd}(\mathsf{app}(x,\mathsf{u}), \mathsf{app}(y,\mathsf{u})) = \mathsf{0}$, whence $T(x) \to T(x*y)$, i.e. $x * y \sqsubseteq x$. In a similar way, we prove $x * y \sqsubseteq y$. Finally, from $T(x) \to T(v)$ and $T(y) \to T(v)$, we deduce $T(x*y) \to T(v)$, observing that $\vdash_{\mathcal{IPOR}^\lambda} T(x*y) \to T(y)$. Notice that the formula $T(x)$ is *not* in Σ^b_1, as its existential quantifier is not bounded.

▶ **Definition 10.** For any formula A of $\mathcal{IPOR}^\lambda$ and fresh variable x, we define $x \Vdash A$:

$$\begin{aligned} x \Vdash A &:= A \vee T(x) \quad (A \text{ atomic}) \\ x \Vdash B \triangleright C &:= x \Vdash B \triangleright x \Vdash C \\ x \Vdash B \to C &:= \forall y(y \vDash B \to x * y \vDash X) \\ x \Vdash \mathcal{Q}y.B &:= \mathcal{Q}y.x \Vdash B. \end{aligned}$$

where $\triangleright \in \{\vee, \wedge\}$, and $\mathcal{Q} \in \{\exists, \forall\}$.

The following Lemma 19 is proved by induction on the structure of formulas, together with a series of intermediate results.

▶ **Lemma 19.** *If $\vdash_{\mathcal{IPOR}^\lambda} A$ without using* **NP***-induction, then $\vdash_{\mathcal{IPOR}^\lambda} x \vDash A$.*

▶ **Lemma 20.** *Let $A = \exists x \preceq \mathsf{t}.B$, where $B \in \Sigma^b_0$. Then, there is a term $\mathsf{u} : s$, with $FV(\mathsf{u}_A)=FV(B)$ such that $\vdash_{\mathcal{IPOR}^\lambda} A \leftrightarrow T(\mathsf{u}_A)$.*

From which we deduce the following three properties for any $A \in \Sigma^b_1$:
i. $\vdash_{\mathcal{IPOR}^\lambda} (x \Vdash A) \leftrightarrow (A \vee T(x))$
ii. $\vdash_{\mathcal{IPOR}^\lambda} (x \Vdash \neg A) \leftrightarrow (A \to T(x))$
iii. $\vdash_{\mathcal{IPOR}^\lambda} (x \Vdash \neg\neg A) \leftrightarrow (A \vee T(x))$,

▶ **Corollary 5** (Markov). If A is a Σ^b_1-formula, then $\vdash_{\mathcal{IPOR}^\lambda} x \Vdash \neg\neg A \to A$.

Finally, we define the extension $(\mathcal{IPOR}^\lambda)^*$ of $\mathcal{IPOR}$, using PIND.

▶ **Definition 11** (PIND). Let PIND(A) indicate the formula:

$$(A(\epsilon) \wedge (\forall x.(A(x) \to A(x\mathsf{0})) \wedge \forall x.(A(x) \to A(x\mathsf{1})) \to \forall x.A(x).$$

Observe that if $A(x)$ is in the form $\exists y \preceq \mathsf{t}.\mathsf{u} = \mathsf{v}$, then the formula $z \Vdash \mathrm{PIND}(A)$ is of the form $\mathrm{PIND}(A(x) \vee T(z))$, which is *not* an instance of the **NP**-induction schema (as $T(z) = \exists x.\mathsf{B}(\mathsf{app}(z,x)) = \mathsf{0}$ is not bounded).

▶ **Definition 12** (Theory $(\mathcal{IPOR}^\lambda)^*$)**.** Let $(\mathcal{IPOR}^\lambda)^*$ indicate the theory extending $\mathcal{IPOR}^\lambda$ with all instances of the induction schema $\mathrm{PIND}(A(x) \vee B)$, where $A(x)$ is of the form $\exists y \preceq \mathsf{t}.\mathsf{u} = \mathsf{v}$ and B is an arbitrary formula with $x \notin \mathrm{FV}(B)$.

▶ **Proposition 5.** For any Σ^b_1-formula A, if $\vdash_{\mathcal{IPOR}^\lambda} A$, then $\vdash_{(\mathcal{IPOR}^\lambda)^*} x \Vdash A$.

We can also extend the realizability interpretation to $(\mathcal{IPOR}^\lambda)^*$ by simply constructing a realizer for $\mathrm{PIND}(A(x) \vee B)$.

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

So, by Theorem 18.i, we observe that for any Σ^b_1-formula A and B with $x \notin \mathrm{FV}(A)$, $\vdash_{\mathcal{IPOR}^\lambda}$ $\mathrm{PIND}(A(x) \vee B)$.

▶ **Corollary 6.** For $A \in \Sigma^b_1$, if $\vdash_{\mathcal{IPOR}^\lambda+EM} \forall x.\exists y \preceq \mathsf{t}.A(x,y)$, then $\vdash_{\mathcal{IPOR}^\lambda} \forall x.\exists y \preceq \mathsf{t}.A(x,y)$.

We conclude by proving the Prop. 6 below:

▶ **Proposition 6.** Let $\forall x.\exists y \preceq \mathsf{t}.A(x,y)$ be a closed theorem of $\mathcal{IPOR}^\lambda$ + (Markov), with $A \in \Sigma^b_1$. Then, there is a $\mathsf{t} : s \Rightarrow s$ of $\mathcal{POR}^\lambda$ such that $\vdash_{\mathcal{IPOR}^\lambda} \forall x.A(x, \mathsf{t}x)$.

Proof. If $\mathcal{IPOR}^\lambda$ + (Markov) proves $\forall x.\exists y.A(x,y)$, then by Prop 1 it also proves $\exists y \preceq \mathsf{t}.A(x,y)$ and $\vdash_{(\mathcal{IPOR})^*} z \Vdash \exists y \preceq \mathsf{t}.A(x,y)$. Let $B := \exists y \preceq \mathsf{t}.A(x,y)$ and $\mathsf{z} = \mathsf{u}_C$, by Lemma 21, $\vdash_{(\mathcal{IPOR}^\lambda)^*} B$ and so, by Lemma 19 and 20, we conclude that there are $\mathbf{t}, \mathbf{u}$ such that $\vdash_{\mathcal{IPOR}^\lambda} \mathbf{t}, \mathbf{u}ⓇB$, which implies $\vdash_{\mathcal{IPOR}^\lambda} A(x, \mathbf{t}x)$. Thus, $\vdash_{\mathcal{IPOR}^\lambda} \forall x.A(x), \mathbf{t}x$. ◀

So by Prop 4, if $\vdash_{\mathcal{IPOR}^\lambda+EM} \forall x.\exists \preceq \mathsf{t}.A(x,y)$, where A is a closed Σ^b_1-formula, then there is a closed term $\mathsf{t} : s \Rightarrow s$ of $\mathcal{POR}^\lambda$ such that $\vdash_{\mathcal{IPOR}^\lambda} \forall x.A(x, \mathsf{t}x)$. As for Corollary 4,

▶ **Corollary 7.** Let $RS^1_2 \vdash \forall x.\exists y \preceq \mathsf{t}.A(x,y)$, where A is a Σ^b_1-formula with only x, y free. For any function $f : \mathbb{S} \times \mathbb{O} \to \mathbb{S}$, if $\forall x.\exists y \preceq \mathsf{t}.A(x,y)$ represents f so that (1) $RS^1_2 \vdash \forall x.\exists! y.A(x,y)$, (2) $[\![A(\overline{\overline{s_1}}, \overline{\overline{s_2}}]\!] = \{\omega \mid f(s_1, \omega) = s_2\}$, then $f \in \mathcal{POR}$.

B Proofs from Section 4.2

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

In order to establish the results in section 4.2, we fix some definitions. We start with those of STMs and their configurations:

▶ **Definition 13** (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 $\circledast \notin \Sigma$.

▶ **Definition 14** (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 15** (Stream Machine Reachability Functions). Given an STM M with transition function δ, we denote with $\vdash_\delta$ its standard step function and we call $\{\triangleright^n_M\}_n$ the smallest family of relations for which:

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

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

▶ **Definition 16** (STM Computation). Given an STM, $M = \langle \mathcal{Q}, q_0, \Sigma, \delta \rangle$, $\eta : \mathbb{N} \longrightarrow \mathbb{B}$ and a function $g : \mathbb{N} \longrightarrow \mathbb{B}$, we say that M *computes* g, written $f_M = g$ 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:

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

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

Similar notations are employed for all the families of Turing-like machines we defined in this paper. However, PTMs require us to extend this definition to probability distributions over $\mathbb{S}$:

▶ **Definition 17** (Sequence of Random Variables associated to a Probabilistic Turing Machine). Given a PTM M, a configuration $\langle \sigma, q, \tau \rangle$, we define 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 $X^{\langle \sigma, q, \tau \rangle}_{M,n}$ describes the configuration reached by the machine after exactly n transitions. We say that a PTM M computes $Y_{M,\sigma}$ iff $\exists t \in \mathbb{N}.\forall \sigma.X^{\langle \sigma, q_0, \tau \rangle}_{M,t}$ is final. In such case $Y_{M,\sigma}$ is the longest suffix of $\pi_1(X^{\langle \sigma, q_0, \epsilon \rangle}_{M,t})$, which does not contain $\circledast$.

We start with the proof of Proposition 8, which establishes the equivalence between STMs and PTMs.

▶ **Proposition 8** (Equivalence of PTMs and STMs). *For any polytime STM ν there exists a polytime PTM ν^* such that for all string $\sigma, \tau \in \mathbb{S}$, $\mu(\{\eta \mid \nu(\sigma, \eta) = \tau\}) = \Pr[\nu^*(\sigma) = \tau]$, and conversely.*

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 $I : \text{STMs} \longrightarrow \text{PTM}$ such that:

$$\forall n \in \mathbb{N}.\{\eta \in \mathbb{B}^{\mathbb{N}} \mid \langle \sigma, q_0, \tau, \eta\rangle \triangleright_{\delta}^{n} \langle \tau, q, \psi, n\rangle\} = \{\eta \in \mathbb{B}^{\mathbb{N}} \mid X_{I(N),n}^{\langle \epsilon, q_0, \sigma\rangle}(\eta) = \langle \tau, q, \psi\rangle\} \tag{1}$$

which entails:

$$\{\eta \in \mathbb{B}^{\mathbb{N}} \mid N(\sigma, \eta) = \tau\} = \{\eta \in \mathbb{B}^{\mathbb{N}} \mid Y_{I(N),\sigma}(\eta) = \tau\}. \tag{2}$$

For this reason, it suffices to construct I and prove that (1) 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. We prove equation (1) by induction on the number of steps required by N to compute its output value.

0 In this case we know that:

$$\{\eta \in \mathbb{B}^{\mathbb{N}} \mid \langle \sigma, q_0, \tau, \eta\rangle \triangleright_{\delta}^{0} \langle \epsilon, q_0, \sigma, \eta\rangle\} = \mathbb{B}^{\mathbb{N}} = \{\eta \in \mathbb{B}^{\mathbb{N}} \mid X_{I(N),0}^{\langle \epsilon, q_0, \sigma\rangle}(\eta) = \langle \epsilon, q, \sigma\rangle\},$$

which proves the claim.

$n+1$ In this case we must show that:

$$\{\eta \in \mathbb{B}^{\mathbb{N}} \mid \langle \sigma, q_0, \tau, \eta\rangle \triangleright_{\delta}^{n+1} \langle \tau, q, \psi, \eta'\rangle\} = \{\eta \in \mathbb{B}^{\mathbb{N}} \mid X_{I(N),n+1}^{\langle \epsilon, q_0, \sigma\rangle}(\eta) = \langle \tau, q, \psi\rangle\},$$

which proves the claim. We also know that:

$$\forall m \leq n.\{\eta \in \mathbb{B}^{\mathbb{N}} \mid \langle \sigma, q_0, \tau, \eta\rangle \triangleright_{\delta}^{m} \langle \tau, q, \psi, \eta''\rangle\} = \{\eta \in \mathbb{B}^{\mathbb{N}} \mid X_{I(N),m}^{\langle \epsilon, q_0, \sigma\rangle}(\eta) = \langle \tau, q, \psi\rangle\},$$

thus, it is easy to show that $\{\eta \in \mathbb{B}^{\mathbb{N}} \mid \langle \sigma, q_0, \tau, \eta\rangle \triangleright_{\delta}^{n+1} \langle \tau, q, \psi, \eta'\rangle\} =$

$$\begin{gathered}\{\eta \in \mathbb{B}^{\mathbb{N}} \mid \langle \sigma, q_0, \tau, \eta\rangle \triangleright_{\delta}^{n} \langle \tau', q', \psi', \mathbf{0}\eta'\rangle \vdash_{\delta} \langle \tau, q, \psi, \eta'\rangle\} \\ \cup \\ \{\eta \in \mathbb{B}^{\mathbb{N}} \mid \langle \sigma, q_0, \tau, \eta\rangle \triangleright_{\delta}^{n} \langle \tau', q', \psi', \mathbf{1}\eta'\rangle \vdash_{\delta} \langle \tau, q, \psi, \eta'\rangle\}.\end{gathered} \tag{3}$$

Concerning $\{\eta \in \mathbb{B}^{\mathbb{N}} \mid X_{I(N),n+1}^{\langle \epsilon, q_0, \sigma\rangle}(\eta) = \langle \tau, q, \psi\rangle\}$, it is equal to

$$\begin{gathered}\{\eta \in \mathbb{B}^{\mathbb{N}} \mid X_{I(N),n}^{\langle \epsilon, q_0, \sigma\rangle}(\eta) = \langle \tau', q', \psi'\rangle \wedge \eta(n) = 0 \wedge \Delta_0(\delta)(\langle \tau', q', \psi'\rangle) = \langle \tau, q, \psi\rangle\} \\ \cup \\ \{\eta \in \mathbb{B}^{\mathbb{N}} \mid X_{I(N),n}^{\langle \epsilon, q_0, \sigma\rangle}(\eta) = \langle \tau', q', \psi'\rangle \wedge \eta(n) = 1 \wedge \Delta_1(\delta)(\langle \tau', q', \psi'\rangle) = \langle \tau, q, \psi\rangle\}.\end{gathered} \tag{4}$$

It is easy to see that the sets in (3) and (4) are pairwise equal thanks to the IH and the definition of I. Claim (2) is a consequence of the definition of Y. Both the machines require the same number of steps. For this reason, if the first is polytime also the second one is so. The opposite direction can be shown in a similar way, using I^{-1} instead of I. ◀

As we mentioned above, to prove Proposition 10 we show the correspondance between the class of STMs and the class of *finite-stream* STMs.

▶ **Lemma 22.** *For each $f \in \mathbf{SFP}$ with time-bound $p \in \mathsf{POLY}$, there is an $h \in \mathcal{PTF}$ such that for any $\eta \in \mathbb{B}^{\mathbb{N}}$ and $x, y \in \mathbb{S}$,*

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

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

Figure 2 Behavior of the function *step*.

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

$$\langle \boldsymbol{\epsilon}, q_0', x, y \rangle \triangleright_M^k \langle \sigma, q, \tau, y' \rangle \quad \Leftrightarrow \quad \langle \boldsymbol{\epsilon}, q_0', x, y\eta \rangle \triangleright_N^k \langle \sigma, q, \tau, y'\eta \rangle.$$

Moreover, N requires a number of steps which is exactly equal to the number of steps required by M, and thus is in $\mathcal{PTF}$, too. We conclude the proof defining $h = f_N$. ◀

The next step is to show that each function $f \in \mathcal{PTF}$ 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 which is equivalent

▶ **Lemma 23.** *For any $f \in \mathcal{PTF}$ 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. A self-contained and comprehensive proof of this result can be found in [17]. 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 $step \in \mathcal{POR}$ which satisfies the simulation schema of Figure 2. 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 $m := \lambda z, x, \omega . f^{|z|}(x, \omega)$ 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 $|step(\sigma, \omega)|$ is $O(|\sigma|)$.
4. If $c := e_c(\sigma, q, \tau, y, \omega)$ for some omega, then there is a function *dectape* such that $\forall \omega \in \mathbb{O} . dectape(x, \omega)$ 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 $sa_{f,t} : \mathbb{S}^{k+2} \times \mathbb{O} \longrightarrow \mathbb{S}$ such that:*

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

Moreover, if f is defined without recurring to Q, m can be defined without Q as well.

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

$$sa_{f,t}(x, \boldsymbol{\epsilon}, \vec{z}, \omega) := x$$
$$sa_{f,t}(x, y\mathbf{b}, \vec{z}, \omega) := f\big(sa_{f,t}(x, y, \omega), \vec{z}, \omega\big)|_t$$

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

- If $n = \boldsymbol{\epsilon}$, $sa_{f,t}$ reduces to $x = f^0(x, \vec{z}, \omega)$.
- If $n = s\mathbf{b}$, the result can be shown applying IH and the definition of $sa'_{f,t}$ to the claim.

◀

Combining these results, we are able to prove Lemma 23

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

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

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:

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

◀

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 8.** 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 $x \in \mathbb{S}$,

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

Proof. Assume $f \in \mathbf{SFP}$ and $y = \eta_{p(|x|)}$. By Lemma 22, there is a function $h \in \mathcal{PTF}$ such that, for any $\eta : \mathbb{N} \longrightarrow \mathbb{B}$ and $x \in \mathbb{S}$,

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

Moreover, due to Lemma 23, there is also a $g \in \mathcal{POR}$ such that for every $x, y \in \mathbb{S}, \omega \in \mathbb{O}$,

$$g(x, y, \omega) = h\big(x, y\big).$$

Then, the desired function is g. ◀

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 18.** 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 $n + 1$.

▶ **Definition 19.** Let us define an auxiliary function $binsucc : \mathbb{S} \times \mathbb{O} \longrightarrow \mathbb{S}$,

$$binsucc(\boldsymbol{\epsilon}, \omega) := \mathbf{1};$$
$$binsucc(x\mathbf{0}, \omega) := x\mathbf{1}|_{x\mathbf{00}};$$
$$binsucc(x\mathbf{1}, \omega) := binsucc(x, \omega)\mathbf{0}|_{x\mathbf{00}}.$$

Then, $bin : \mathbb{S} \times \mathbb{O} \longrightarrow \mathbb{S}$ is defined as:

$$bin(\boldsymbol{\epsilon}, \omega) := \mathbf{0};$$
$$bin(x\mathbf{b}, \omega) := binsucc\big(bin(x, \omega), \omega\big)|_{x\mathbf{b}}.$$

We call $dy : \mathbb{S} \times \mathbb{O} \longrightarrow \mathbb{S}$ the function of $\mathcal{POR}$ defined as follows:

$$dy(x, \omega) := lrs(bin(x, \omega), \omega),$$

where lrs is a string manipulator, which removes the left-most bit from a string if it exists, otherwise it returns $\boldsymbol{\epsilon}$.

▶ **Lemma 25.** *The function $dyad(n)$ is bijective.*

Proof. This function is an injection since different numbers have different binary encodings and for any $n > 0 \in \mathbb{N}$ and $\omega \in \mathbb{O}$, $bin(n, \omega)$ has $\mathbf{1}$ as leftmost bit (as provable by induction on n, leveraging the definition of bin). So, if we take two distinct binary encodings of natural numbers n, m and call them $\mathbf{1}\sigma$ and $\mathbf{1}\tau$, $\sigma \neq \tau$ must hold (otherwise $n = m$, as the function which associates numbers to their binary representation is itself a bijection).

The function is surjective. We know that dy is computed removing a bit which is always $\mathbf{1}$. This entails that each string $\sigma \in \mathbb{S}$ is the image of the natural number n such that the binary encoding of $n + 1$ is $\mathbf{1}\sigma$. This number always exist. ◀

▶ **Proposition 26.** *For any $n \in \mathbb{N}, \sigma \in \mathbb{S}$ and $\omega \in \mathbb{O}$, $|\sigma| = n + 1 \rightarrow dy(\sigma, \omega) = dyad(n)$.*

Proof Sketch. By induction on n. ◀

▶ **Definition 20.** 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 27** (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 formers 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 Proposition 26). For smaller values of j, the first claim is a consequence of the definition of e together with IH.

◀

▶ **Definition 21.** 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 28.** *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. From Proposition 26 and Lemma 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 being different. Then, there is a $\sigma \in \mathbb{S}$, such that $\omega_1(\sigma) \neq \omega_2(\sigma)$ and, by Proposition 26, the value of ω does not affect the value of its output, moreover $dy(\mathbf{1}^{\cdot+1}, \omega)$ it is a bijection so there is an $n \in \mathbb{N}$, such that $dy(\mathbf{1}^{n+1}, \omega) = \sigma$, so we get $\eta(n) = \omega_1(\sigma) \neq \omega_2(\sigma) = \eta(n)$, which is a contradiction. ◀

▶ **Corollary 9.** The relation $\sim_{dy}$ is a bijection.

Proof. Consequence of Lemma 28. ◀

▶ **Lemma 29.**

$$\eta \sim_{dy} \omega \to \forall n \in \mathbb{N}.\eta_n = e(\underline{n}_{\mathbb{N}}, \omega).$$

Proof. By contraposition: suppose $\eta_n \neq e(\underline{n}_{\mathbb{N}}, \omega)$. As a consequence of the correctness of e (Lemma 27), 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 10.

Proof of Proposition 10. From Corollary 8, we know that there is a function $f' \in \mathcal{POR}$, and a $p \in \mathsf{POLY}$ such that:

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

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 $\mathbf{1}^{|x|+1}$. Indeed, by Lemma 29, it holds that $\overline{\eta}_{p(x)} = e(p(size(x, \omega), \omega)$, 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 $\overline{\omega} \in \{\omega \in \mathbb{O} \mid f'(x, e(p'(size(x, \omega), \omega), \omega), \omega) = y\}$, 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: $\mu\big(\{\eta \in \mathbb{B}^{\mathbb{N}} \mid f(x, \eta) = y\}\big) = \mu\big(\{\omega \in \mathbb{O} \mid f'(x, e(p(size(x, \omega), \omega), \omega), \omega) = y\}\big)$, 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 22** (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*.

▶ **Definition 23** (Store)**.** A store is a function $\Sigma : \mathsf{Id} \rightharpoonup \{\mathsf{0},\mathsf{1}\}^*$, an *empty* store is a store which is total and constant on ϵ. We represent such object as $[]$.

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 24** (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:

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

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

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

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

▶ **Definition 25** (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 26** (Function evaluated by a $\mathbf{SIFP_{RA}}$ program)**.** We say that the function evaluated by a correct $\mathbf{SIFP_{RA}}$ program P is $[\![\cdot]\!] : \mathcal{L}(Stm_{\mathbf{RA}}) \longrightarrow (\mathbb{S}^n \times \mathbb{O} \longrightarrow \mathbb{S})$, defined as below[2]:

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

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:

[2] Instead of the infixed 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

▶ **Lemma 30** (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 $f \in \mathcal{POR}$, then P does not contain any $\texttt{Flip}(e)$ statement.*

The proof of this result is quite simple under a technical point of view, because 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 30. 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: $\mathfrak{L}_E := R \leftarrow \epsilon$; $\mathfrak{L}_{S_0} := R \leftarrow X_0.\mathtt{0}$; $\mathfrak{L}_{S_1} := R \leftarrow X_0.\mathtt{1}$; $\mathfrak{L}_{P^n_i} := R \leftarrow X_i$; $\mathfrak{L}_C := R \leftarrow X_1 \sqsubseteq X_2$; $\mathfrak{L}_Q := \texttt{Flip}(X_1)$. The correctness of these base cases is trivial. Moreover, it is simple to see that the only translation containing $\texttt{Flip}(e)$ for some $e \in \mathcal{L}(\mathsf{Exp})$ 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. For an extensive and self-contained proof of this result, the reader can refer to [17]. ◀

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 31.

▶ **Lemma 31.** *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 entering in the details of this Lemma, we must deifine the language $\mathbf{SIFP_{LA}}$ together with its standard semantics:

▶ **Definition 27** ($\mathbf{SIFP_{LA}}$). 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:

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

▶ **Definition 28** (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:

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

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

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

In particular, we prove Lemma 31 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 an induction proof of the reduction from $\mathbf{SIFP_{RA}}$ to $\mathbf{SIFP_{LA}}$, in particular we employ a list Ψ containing pairs (x, b). This list is meant to keep track of the previous call to the primitive $\texttt{Flip}(x)$ on the side of $\mathbf{SIFP_{RA}}$, and of the result of the x-th call of the primitive $\texttt{RandBit}()$ on the side of $\mathbf{SIFP_{LA}}$.

This was done because the main issue we encountered proving this lemma was the simulation of the access to the random tape. To achieve this, we defined the translation in such a way that it stores, in a specific and fresh register, an associative table recording all the queries σ within a $\texttt{Flip}(\sigma)$ instruction and the result b which was picked from η and returned. The addition of the map Ψ allows to replicate the content of the associative table Ψ explicitly the in the program's semantics. This simulation requires a translation of the $\texttt{Flip}(e)$ instructions into an equivalent procedure. In particular, these statements are simulated as follows:

- At each simulated query $\texttt{Flip}(e)$, 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 $\texttt{Flip}(e)$ to a call of $\texttt{RandBit}()$ which outputs either $\texttt{b} = \mathbf{0}$ or $\texttt{b} = \mathbf{1}$.
  - It records the couple $\langle e, \texttt{b}\rangle$ in the associative table and returns $\texttt{b}$.

Even in this case the construction 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. The formal proof of Lemma 31 is given in its entirety in [17]. It is structured as follows:

- We show that the *big-step* and *small-step* operational semantics given for $\mathbf{SIFP_{RA}}$ and $\mathbf{SIFP_{LA}}$ are equally expressive.
- We define a relation Θ between configurations of the *small-step* semantics of $\mathbf{SIFP_{RA}}$ and $\mathbf{SIFP_{LA}}$. In particular, Θ is defined upon three co-operating entities:
  - A function
  $$\alpha : \mathcal{L}(\mathsf{Stm}_{\mathbf{RA}}) \longrightarrow \mathcal{L}(\mathsf{Stm}_{\mathbf{LA}})$$
  which maps the program $P_{\mathbf{RA}} \in \mathcal{L}(\mathsf{Stm}_{\mathbf{RA}})$ into its corresponding $P_{\mathbf{LA}} \in \mathcal{L}(\mathsf{Stm}_{\mathbf{LA}})$ translating the $\texttt{Flip}(e)$ instruction as we described above.
  - A relation β between a triple $\langle P_{\mathbf{RA}}, \Sigma, \Psi\rangle$ and a single store Γ. This relation is meant to capture the configuration-to-store dependencies between the configuration of the $\mathbf{SIFP_{RA}}$ program $P_{\mathbf{RA}}$ running with the store Σ and computed associative table Ψ and the store Γ of the simulating program $P_{\mathbf{LA}}$. This constraint basically requires Γ to store a representation of Ψ into a dedicated register.
  - A function γ which transforms the constraints on the oracle gathered by the relation $\rightsquigarrow_{\mathbf{RA}}$ to the information collected by the relation $\rightsquigarrow_{\mathbf{RA}}$.

  Thus Θ is defined as follows:

  ▶ **Definition 29** (Θ Relation).
  $\Theta\left(\langle P, \Sigma_1, \Psi\rangle, \langle Q, \Sigma_2, \Phi\rangle\right)$ holds iff:
  - $\alpha(P) = Q$;
  - $\beta(\langle P, \Sigma_1, \Psi\rangle, \Sigma_2)$;
  - $\gamma(\Psi) = \Phi$;
  - $\mu(\Psi) = \mu(\Phi)$.

$$\begin{tikzcd}
\langle P_{\mathbf{RA}}; Q_{\mathbf{RA}}, \Sigma, \Psi\rangle \arrow[r, rightsquigarrow] \arrow[d, "\Theta"] & \langle Q_{\mathbf{RA}}, \Sigma', \Psi'\rangle \arrow[d, "\Theta"] \\
\langle P_{\mathbf{LA}}; Q_{\mathbf{LA}}, \Gamma, \Phi\rangle \arrow[r, rightsquigarrow, "*"] & \langle Q_{\mathbf{LA}}, \Gamma', \Phi'\rangle
\end{tikzcd}$$

Figure 3 Commutation schema between $\mathbf{SIFP_{RA}}$ and $\mathbf{SIFP_{LA}}$

- We prove a simulation result with respect to the Θ relation. In other words, we show that Θ associates to each triple $\langle P_{\mathbf{RA}}, \Sigma, \Psi\rangle$ other triples $\langle P_{\mathbf{LA}}, \Gamma, \Phi\rangle$ which weakly simulate the relation $\rightsquigarrow_{\mathbf{RA}}$ with respect to $\rightsquigarrow_{\mathbf{LA}}$. This is depicted by Figure 3.

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*.

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 7.** 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 invariants 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, $q^I_{s_i}, q^1_{s_i}, q^2_{s_i}, \ldots, q^F_{s_i}$.

- 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 `while()`{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 `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 invariants 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 32.** *For every $M = \langle \mathcal{Q}, \Sigma, \delta, q_0\rangle \in \mathbf{SFP_{OD}}$, the machine $N = \langle \mathcal{Q}, \Sigma, H(\delta), q_0\rangle \in$ **SFP** is such that for every $n \in \mathbb{N}$, for every configuration of M $\langle \sigma, q, \tau, \eta\rangle$ 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 M it is possible to build a stream machine N which is equivalent to M. Intuitively, the encoding from an *on-demand* stream machine M to an ordinary stream machine takes the transition function δ of M and substitutes each transition not causing the oracle tape to shift — i.e. tagged with $\natural$ — with two distinct transitions, with respectively `0` and `1` instead of the symbol $\natural$. This causes the resulting machine to produce an identical transition but shifting the head on the oracle tape on the right.

▶ **Definition 30** (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 32. 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:

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

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

► **Proposition 33** (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 30, Lemma 31, Proposition 7 and Lemma 32. ◀