A Recursion-Theoretic Characterization of the Probabilistic Class PP

Ugo Dal Lago, Reinhard Kahle, Isabel Oitavem

A Recursion-Theoretic Characterization of the Probabilistic Class PP

Ugo Dal Lago
University of Bologna, Italy
INRIA Sophia Antipolis, France

Reinhard Kahle
Carl Friedrich von Weizsäcker Center, Universität Tübingen, Germany
Center for Mathematics and Applications (CMA), FCT, Nova University Lisbon, Caparica, Portugal

Isabel Oitavem[1]
Center for Mathematics and Applications (CMA), FCT, Nova University Lisbon, Caparica, Portugal
Department of Mathematics, FCT, Nova University Lisbon, Caparica, Portugal

Abstract

Probabilistic complexity classes, despite capturing the notion of feasibility, have escaped any treatment by the tools of so-called implicit-complexity. Their inherently semantic nature is of course a barrier to the characterization of classes like BPP or ZPP, but not all classes are semantic. In this paper, we introduce a recursion-theoretic characterization of the probabilistic class PP, using recursion schemata with pointers.

2012 ACM Subject Classification Theory of computation → Recursive functions

Keywords and phrases Implicit complexity, tree-recursion, probabilistic classes, polynomial time, PP

Digital Object Identifier 10.4230/LIPIcs.MFCS.2021.35

Funding *Ugo Dal Lago*: Research supported by the ERC CoG DIAPASoN, GA 818616, and by the ANR Project PPS 19CE480014.
Reinhard Kahle: Research supported by the Udo Keller Foundation and as for Isabel Oitavem.
Isabel Oitavem: National funds throught the FCT – Fundação para a Ciência e Tecnologia, I.P., under the scope of the project UIDB/00297/2020 (Center for Mathematics and Applications).

1 Introduction

Implicit computational complexity aims at introducing and studying characterizations of complexity classes by recursion theory, proof theory, and programming language tools. This has allowed to shed some light on the impact of programming and recursion schemes to resource consumption. As an example, nested recursion on notation is known to be harmful to time complexity, and thus needs to be appropriately controlled if one wants to stay within the realm of polynomial time computable functions, as shown by Bellantoni-Cook and Leivant in their works on safe and tiered recursion [2, 9].

If one wants to go beyond polynomial time, a way of enriching recursion schemes without breaking the correspondence with (relatively small) complexity classes is the use of recursion schemes based on *pointers* [3], which leads to characterizations of NP and FPSPACE (the class of *functions* corresponding to PSPACE), [14, 13].

One family of complexity classes between polytime and polyspace which has so far escaped any implicit treatment are the probabilistic ones. Our goal in this paper is precisely the one of exploring the potential of pointers in recursion-theoretic contexts as a tool to characterize

[1] corresponding author

probabilistic classes of computational complexity. In this work we study PP, the class of decision problems solvable by probabilistic Turing machines in polynomial time with an error probability of less than $\frac{1}{2}$ for all instances. The class PP was originally defined by Gill in [6]. It is well-known that PP contains NP and that it is contained in PSPACE; it is open whether these inclusions are proper or not.

Our strategy consists in extending the existing recursion scheme for NP going towards the one for FPSPACE, but without going too far. Surprisingly, this is possible despite the inherently quantitative nature of PP's acceptance criterion. This way, we obtain the first purely recursion-theoretic characterization of the probabilistic class PP.

This characterization is described in two stages, $\mathbf{ST}_{\mathsf{P}}$ and $\mathbf{ST}_{\mathsf{PP}}$, where $\mathbf{ST}_{\mathsf{P}}$ characterizes the class of functions computable in polynomial time by deterministic Turing machines, FPTIME – see [2]. $\mathbf{ST}_{\mathsf{PP}}$ results then from "strengthening" $\mathbf{ST}_{\mathsf{P}}$ with a scheme designed to characterize the decision problems of PP. That scheme is the tree-recursion scheme of FPSPACE [13], but with a fixed step function. Therefore, the characterization of the class PP given here is aligned with the existing recursion-theoretic characterizations of FPTIME and FPSPACE.

2 Algebras, Functions, and Complexity Classes

In this section, we introduce some preliminary concepts that will accompany us in the rest of the paper. In particular, we will show how binary strings and functions over them allow us to capture standard, deterministic classes like FPTIME and FPSPACE.

2.1 Algebras and Recursion Schemes

Let us consider the word algebra $\mathbb{W}$, i.e. the algebra generated by one nullary and two unary constructors, respectively indicated as ϵ, $\mathbf{S}_0$ and $\mathbf{S}_1$. The algebra $\mathbb{W}$ can naturally be interpreted over the set $\{0,1\}^*$ of all binary words. We abbreviate $\mathbf{S}_0x$ and $\mathbf{S}_1x$ as $x0$ and $x1$, respectively. We consider a predecessor symbol $\mathbf{P}$ of arity 1, and a conditional function symbol $\mathbf{C}$ of arity 4. They are defined as follows: $\mathbf{P}(\epsilon) = \epsilon$, $\mathbf{P}(xi) = x$ and $\mathbf{C}(\epsilon, x, y_0, y_1) = x$, $\mathbf{C}(zi, x, y_0, y_1) = y_i$, $i \in \{0,1\}$.

Recursion schemes on the algebra $\mathbb{W}$ can be built in two different ways:

1. First of all, one can proceed by *recursion on notation*, namely by building a function f out of g and h by stipulating that $f(\epsilon, \bar{x}) = g(\epsilon, \bar{x})$ and $f(zi, \bar{x}) = h(zi, \bar{x}, f(z, \bar{x}))$, where $i \in \{0,1\}$;
2. Secondly, one can also build f through *tree-recursion with pointers* from g and h, namely by $f(p, \epsilon, \bar{x}) = g(p, \epsilon, \bar{x})$ and $f(p, zi, \bar{x}) = h(p, zi, \bar{x}, f(p0, z, \bar{x}), f(p1, z, \bar{x}))$, $i \in \{0,1\}$.

In both the recursion schemes above, g is designated as the *base function* and h as the *step function*. In tree-recursion, the first input of f is called the *pointer*. Noticeably, the characterization of PP presented here results from considering the scheme (2) with a fixed step function. Informally, the step function that we fix is a FPTIME function corresponding to binary addition (we add the number of accepting configurations), but with some nuances in order to capture the probabilistic class PP.

Above and along the paper, $\bar{x}$ abbreviates $x_1, \cdots, x_n$ for some natural number n. Moreover, $|\bar{x}| = (|x_1|, \cdots, |x_n|)$ where, for $1 \leq k \leq n$, $|x_k|$ denotes the length of x_k, i.e. the number of $\mathbf{S}_0$ and $\mathbf{S}_1$ in x_k. We extend these notations to function symbols.

2.2 Capturing Polynomial Time and Space: ST_{P} and $\mathsf{ST}_{\mathsf{FPSPACE}}$

If either recursion on notation or tree recursion are applied without any restriction, there is no hope to capture interesting complexity classes: in either case we end up capturing the whole class of primitive recursion. From now on, then, we adopt the framework introduced by Bellantoni-Cook [2], which has proved to be quite useful in appropriately restricting function algebras so as to capture small complexity classes. In particular, function terms have two sorts of input positions, dubbed *normal* and *safe*. As is customary, we write normal and safe input positions by this order, separated by semicolon, e.g. in the expression $f(\bar{x};\bar{y})$, the parameters in $\bar{x}$ are normal while those in $\bar{y}$ are safe.

We are now in a position to define the three closure operators which we will employ in the following, and which will be

▶ **Definition 1** (Closure Operators). *Let g, h, $\bar{r}$ and $\bar{s}$ be sorted functions. The following are three ways of building a new sorted function term out of (some of) them.*

- ***Sorted Composition***, SC, *by which we can build the sorted function f such that*

$$f(\bar{x};\bar{y}) = h(\bar{r}(\bar{x};);\bar{s}(\bar{x};\bar{y}))$$

- ***Sorted Recursion on Notation***, *SR, which allows us to derive the sorted function f by a recursion scheme:*

$$\begin{aligned} f(\epsilon,\bar{x};\bar{y}) &= g(\epsilon,\bar{x};\bar{y}) \\ f(z0,\bar{x};\bar{y}) &= h(z0,\bar{x};\bar{y},f(z,\bar{x};\bar{y})) \\ f(z1,\bar{x};\bar{y}) &= h(z1,\bar{x};\bar{y},f(z,\bar{x};\bar{y})). \end{aligned}$$

- ***Sorted Tree-Recursion***, *STR, which derives the sorted function f, again by recursion*

$$\begin{aligned} f(p,\epsilon,\bar{x};\bar{y}) &= g(p,\epsilon,\bar{x};\bar{y}) \\ f(p,z0,\bar{x};\bar{y}) &= h(p,z0,\bar{x};\bar{y},f(p0,z,\bar{x};\bar{y}),f(p1,z,\bar{x};\bar{y})) \\ f(p,z1,\bar{x};\bar{y}) &= h(p,z1,\bar{x};\bar{y},f(p0,z,\bar{x};\bar{y}),f(p1,z,\bar{x};\bar{y})) \end{aligned}$$

A few observations about the two recursion operators are helpful now. Please note that, according to the semi-colon discipline, both the *pointer* p and the *recursion variable* z are in normal input positions, while the results of *recursive calls* go into safe input positions. This, in particular, prevents the results of recursive calls to be fed to a function itself defined by recursion on that same variable, ultimately avoiding a blowup in size and complexity.

▶ **Definition 2** (Function Algebras). *Let $\mathcal{I}$ be the class of function terms including the constructors of $\mathbb{W}$ (i.e. the functions ϵ, $\mathbf{S}_0$, $\mathbf{S}_1$), the predecessor $\mathbf{P}$, the conditional $\mathbf{C}$, and the projection functions (over both input sorts).*

1. $\mathbf{ST}_{P}$ *is the closure of $\mathcal{I}$ under* SC *and SR;*
2. $\mathbf{ST}_{FPSPACE}$ *is the closure of $\mathcal{I}$ under* SC *and STR.*

As known results one has that:

▶ **Proposition 3.**

1. $\mathbf{ST}_{P}$ *characterizes FPTIME and* $\mathbf{ST}_{FPSPACE}$ *characterizes FPSPACE – see, respectively, [2] and [13];*
2. $\mathbf{ST}_{P} \subseteq \mathbf{ST}_{FPSPACE}$, *as classes of input-sorted function terms.*

Notice that in this framework one can run a recursion over an output of a function which is itself defined by recursion. There are some other properties concerning $\mathbf{ST}_{\mathsf{P}}$ that we use in the paper. We summarize them here for further reference – see [2] and [14], Remark 2.

▶ Remark 4.

1. Let $f(\bar{x})$ be a polytime function. Then $f(\bar{x};)$ is in $\mathbf{ST}_{\mathsf{P}}$.
2. For any polynomial (with natural coefficients) q, there exists a term $t \in \mathbf{ST}_{\mathsf{P}}$ such that $\forall \bar{x}\ q(|\bar{x}|) = |t(\bar{x};)|$;
3. For any polytime function F there exist a function term f, in $\mathbf{ST}_{\mathsf{P}}$, and a monotone polynomial q_F such that $\forall \bar{w} \forall y \ \ |y| \geq q_F(|\bar{w}|) \Rightarrow F(\bar{w}) = f(y; \bar{w})$.

3 The Algebra $\mathbf{ST}_{\mathsf{PP}}$

In this section, we define a function algebra $\mathbf{ST}_{\mathsf{PP}}$, based on $\mathbf{ST}_{\mathsf{P}}$ and on the tree-recursion scheme with pointers. The main idea behind $\mathbf{ST}_{\mathsf{PP}}$ is to constrain $\mathbf{ST}_{\mathsf{FPSPACE}}$ in such a way that tree-recursion is restricted to a very specific step function. The next section is devoted to introducing and motivating this function.

3.1 On the $\boxplus$ Function

In the following, we will extensively work with the following FPTIME functions:

- The unary function *read* which, on input w, returns 10 if the last bit of w is 1, and 0 otherwise;
- Addition on binary words *seen as* natural numbers, indicated as $+$. It results from considering the following

$$\begin{array}{ccccccc} 0 & 1 & 10 & 11 & 100 & \ldots & \text{(binary words)} \\ \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & & \\ 0 & 1 & 2 & 3 & 4 & \ldots & \text{(natural numbers)} \end{array}$$

  and the usual addition over $\mathbb{N}$, extending it to all binary words assuming that the empty string ϵ and words starting with 0 all correspond to $0 \in \mathbb{N}$. For instance, $+(10, 10) = 100$, while $+(00, 10) = 10$. We use infix notation for $+$;
- The binary function $\oplus$ such that $\oplus(w, v)$ is $\mathit{read}(w) + \mathit{read}(v)$, for which we also use the infix notation. As an example, $01 \oplus 100 \oplus 1 = \mathit{read}(01) + \mathit{read}(100) + \mathit{read}(1) = 10 + 0 + 10 = 100$. The function $\oplus$ only depends on the last bit of its two arguments. Notice that any finite sum $\sum_n \mathit{read}(w_n)$ is always a binary word ending with 0, because for every such binary word w it holds that $\mathit{read}(w)$ is 0 or 10;
- The binary function $\#$ defined as follows:

$$\#(z, w) = \begin{cases} 1 & \text{if } w > 2^{|z|} \\ 0 & \text{otherwise,} \end{cases}$$

  where $2^{|z|}$ is the binary representation of the natural number $2^{|z|}$, i.e., it holds that $2^{|\epsilon|} = 1$ and that $2^{|zi|} = \mathbf{S}_0(2^{|z|})$.

We reserve the symbol $+$ for the binary addition as described above. Whenever needed, we use $+_{\mathbb{N}}$ to denote the usual addition over the natural numbers. The "greater than" relation defined over binary words ordered by length and, within the same length, lexicographically is indicated as $>$;

Finally, let the function $\boxplus$ be a polytime function satisfying the following equations (where i ranges over $\{0,1\}$):

$$\begin{aligned}
\boxplus(\epsilon, i, w_0, w_1) &= \#(i, w_0 \oplus w_1), \\
\boxplus(p, i, w_0, w_1) &= w_0 \oplus w_1, && \text{if } p \neq \epsilon \\
\boxplus(\epsilon, zi, w_0, w_1) &= \#(zi, w_0 + w_1), && \text{if } z \neq \epsilon, \\
\boxplus(p, zi, w_0, w_1) &= w_0 + w_1, && \text{if } p, z \neq \epsilon,
\end{aligned}$$

The conditions imposed now will become clear later. $\boxplus$ is a polytime function. Therefore Remark 4(1) ensures the existence of a term in $\mathbf{ST}_{\mathsf{P}}$ – let us reuse the symbol $\boxplus$ to denote it – such that $\boxplus(p, z, w_0, w_1;\,) = \boxplus(p, z, w_0, w_1)$. The input-sorted function $\boxplus \in \mathbf{ST}_{\mathsf{P}}$ is used as step function in the tree-recursion scheme.

3.2 The Algebra $\mathbf{ST}_{\mathsf{PP}}$

It is now time to introduce the function algebra which constitutes the main contribution of this paper, and which will be proved to characterize the class PP in the coming section.

▶ **Definition 5.** *$\mathbf{ST}_{PP}$ is the closure of $\mathbf{ST}_P$ under SC_P and $TR[\boxplus]$, where*

- ***Restricted sorted composition**, SC_P, allows to build a sorted function f out of h and $\bar{r}, \bar{s} \in \mathbf{ST}_P$ as follows:*

$$f(\bar{x}; \bar{y}) = h(\bar{r}(\bar{x};\,); \bar{s}(\bar{x}; \bar{y}))$$

- ***Tree-recursion with step function** $\boxplus$ derives the function f out of $g \in \mathbf{ST}_P$ as follows:*

$$\begin{aligned}
f(p, \epsilon, \bar{x};\,) &= g(p, \epsilon, \bar{x};\,) \\
f(p, z0, \bar{x};\,) &= \boxplus(p, z0, f(p0, z, \bar{x};\,), f(p1, z, \bar{x};\,);\,) \\
f(p, z1, \bar{x};\,) &= \boxplus(p, z1, f(p0, z, \bar{x};\,), f(p1, z, \bar{x};\,);\,)
\end{aligned}$$

As already noticed, under this input-sorted discipline recursive calls are usually taken as safe arguments of the step function. However, that becomes irrelevant when the recursion scheme imposes a fixed step function – like in $\mathrm{TR}[\boxplus]$. The definition of any function term in $\mathbf{ST}_{\mathsf{PP}}$ involves at most one application of $\mathrm{TR}[\boxplus]$. This is a consequence of having the base function of $\mathrm{TR}[\boxplus]$ – g – and the inner functions of SC_P – $\bar{r}$ and $\bar{s}$ – all in $\mathbf{ST}_{\mathsf{P}}$.

It is known that $\mathsf{P} \subseteq \mathsf{PP} \subseteq \mathsf{PSPACE}$. On the term system side we observe that the correspondent inclusions are preserved.

▶ Remark 6. $\mathbf{ST}_{\mathsf{P}} \subseteq \mathbf{ST}_{\mathsf{PP}} \subseteq \mathbf{ST}_{\mathsf{FPSPACE}}$, as classes of input-sorted function terms. The first inclusion is obvious, the second makes use of items (2) and (3) of Remark 4.

In the rest of this section, we will explain how the $\mathrm{TR}[\boxplus]$ scheme somehow mimics the acceptance condition underlying PP. Suppose a function is obtained through $\mathrm{TR}[\boxplus]$ from a base function g; then its evaluation on an input $(p, z, \bar{x};\,)$, for $z \neq \epsilon$ gives rise to a tree similar to that in Figure 1, where the pointer p and the recursion input z are omitted. By using them (the first and the second inputs of $\boxplus$), one is able to obtain different outputs for $\boxplus$ depending on the level it occurs in the tree above. Therefore, according to the definition of $\boxplus$, one can distinguish two different situations.

- If f is evaluated on arguments in the form $(\epsilon, z, \bar{x})$, then the tree from Figure 1 becomes the one in Figure 2(a). Again, inputs are omitted. One reads from the leaves, and by performing binary addition at all internal nodes one brings up to the root of the tree the

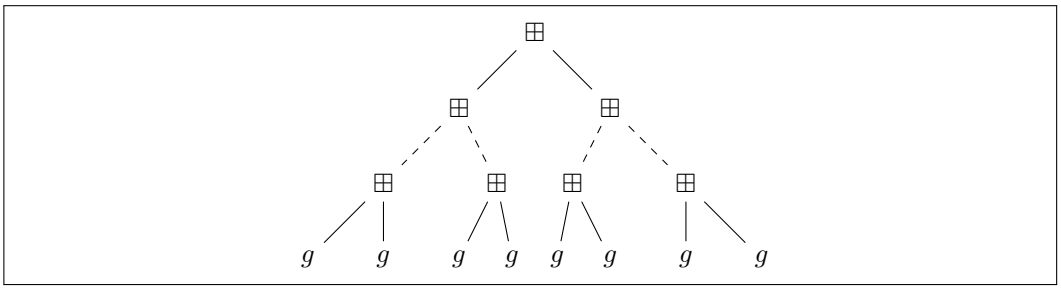
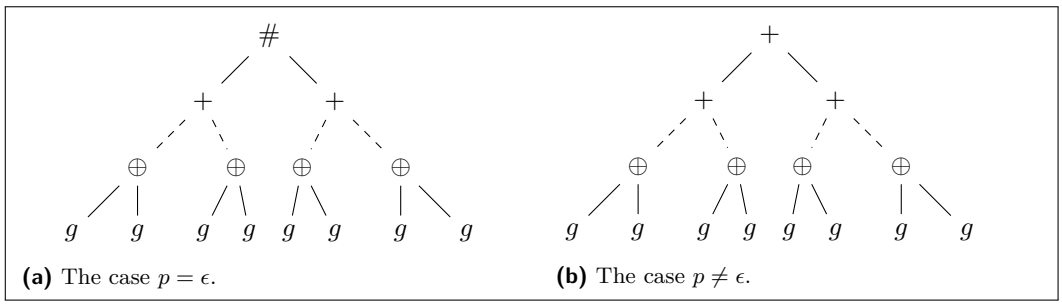


Figure 1 The unfolding of $f(p, z, \bar{x})$.



(a) The case $p = \epsilon$. **(b)** The case $p \neq \epsilon$.

Figure 2 The unfolding of $f(p, z, \bar{x})$, depending on the value of p.

information about how many times 1 occurs at the leaves. Notice that the function #, after performing the binary addition operation, returns 1 if the sum meets the threshold (i.e. if strictly more than half of the leaves are labeled by 1) and 0 otherwise.

- If f is evaluated on arguments in the form $(p, z, \bar{x})$, where $p \neq \epsilon$, then the tree becomes the one in Figure 2(b). The pointer increases along the paths of the tree. So, the first input of ⊞ does not assume the value ϵ. Therefore, the test function # is not called. Recall that ⊕ reads the last bit of the leaves, coding the 1's by 10 (i.e. by the binary representation of 2). It should be clear that all nodes are labeled by strings ending by 0 (because they are binary representation of even numbers). In particular, the value returned to the root of the tree is a 0-1 word ending by 0.

4 $\mathbf{ST}_{\mathsf{PP}}$ Characterizes PP

In this section, we prove that the algebra $\mathbf{ST}_{\mathsf{PP}}$ we introduced in the previous section indeed characterizes our target class, namely PP.

We adopt the definition of PP given in [1]. In particular, we consider non-deterministic Turing machines as the underlying model of computation, and we make the assumption that every step of the computation can be made in *exactly two* possible ways. Thus, in the course of the computation, every configuration of the machine has exactly two next configurations. Machines are "clocked" by some constructible function, and the number of steps in each computation is exactly the number of steps allowed by the clock. If a final state is reached before this number of steps, then the computation is continued, doing nothing up to this number of steps. Moreover, every computation ends in a final state, which can be either ACCEPT or REJECT. A *probabilistic Turing machine* M, PTM for short, is a non-deterministic Turing machine as above in which acceptance is defined quantitatively rather than logically: the input $\bar{x}$ is accepted if, and only if, *more than half* of the computations of M on $\bar{x}$ end in the ACCEPT final state.

PP is the class of boolean functions (or languages) computed (resp. accepted) by polynomially clocked PTMs. While PP is a class of two valued functions, $\mathbf{ST}_{\mathsf{PP}}$ corresponds to a class of functions whose values can range over binary words. As a consequence, our correspondence theorem will say that $\mathcal{B}(\mathbf{ST}_{\mathsf{PP}})$ coincides with PP, where $\mathcal{B}(\mathbf{ST}_{\mathsf{PP}})$ denotes the boolean part of $\mathbf{ST}_{\mathsf{PP}}$.

4.1 The Lower Bound

In this subsection, we will prove that $\mathbf{ST}_{\mathsf{PP}}$ is powerful enough to capture the behavior of any polynomially clocked probabilistic Turing machine. This proves that $\mathbf{ST}_{\mathsf{PP}}$ is at least as expressive as our target complexity class..

▶ **Lemma 7.** *Polynomially clocked PTMs can be simulated by functions in* $\mathbf{ST}_{PP}$.

Proof. Let M be a PTM which is clocked by some polynomial q (on the length of the input). We are going to simulate M by $\mathbf{ST}_{\mathsf{PP}}$ function terms. Let us assume, without any loss of generality, that machine configurations are encoded by binary words so that codes end by the code of the respective state. Codes of accepting final states end by 1 and any other state ends by 0. One may assume that, for a given input $\bar{x}$, all the configuration codes have the same length, $l(|\bar{x}|)$, which is polynomial on $|\bar{x}|$. Notice that all non-terminating configurations have two successor configurations. Therefore, we can split the transition function δ of M into two δ_0 and δ_1. Let c be a $\mathbf{ST}_{\mathsf{PP}}$ function such that $c(\bar{x};)$ is the code of the initial configuration. Similarly, let t be a $\mathbf{ST}_{\mathsf{PP}}$ function such that $|t(\bar{x};)| = q(|\bar{x}|)$. (That t can be defined in $\mathbf{ST}_{\mathsf{PP}}$ come from the fact that q is a polynomial, cf. Remark 4(2)). For $i \in \{0,1\}$, one may consider polytime computable functions Δ_i which, for a given configuration code w, return the next configuration code according to δ_i, or return w itself if there is no next configuration according to δ_i. Thus, by Remark 4(3), there exists a function term $\hat{\Delta}_i$ in $\mathbf{ST}_{\mathsf{P}}$ and a polynomial q_{Δ_i} such that $\forall w\ \forall y\ \ |y| \geq q_{\Delta_i}(|w|) \Rightarrow \Delta_i(w) = \hat{\Delta}_i(y; w)$. Replacing, in the previous expression, y by $L_{\Delta_i}(\bar{x};)$ one has that $\forall w\ \forall \bar{x}\ \ |L_{\Delta_i}(\bar{x};)| \geq q_{\Delta_i}(|w|) \Rightarrow \Delta_i(w) = \hat{\Delta}_i(L_{\Delta_i}(\bar{x};); w)$, where L_{Δ_i} is a $\mathbf{ST}_{\mathsf{P}}$ term as follows. Given an input $\bar{x}$, all configuration codes w satisfy $|w| = l(|\bar{x}|)$ where l is polynomial in $|\bar{x}|$. Thus, $q_{\Delta_i}(|w|)$ is equal to $(q_{\Delta_i} \circ l)(|\bar{x}|)$. The composition of polynomials is a polynomial, and so $q_{\Delta_i} \circ l$ is a polynomial in $|\bar{x}|$. Therefore, by Remark 4(2), there exists a function term L_{Δ_i} in $\mathbf{ST}_{\mathsf{P}}$ such that $|L_{\Delta_i}(\bar{x};)| = (q_{\Delta_i} \circ l)(|\bar{x}|)$. Now, recalling that $q_{\Delta_i}(|w|)$ is $(q_{\Delta_i} \circ l)(|\bar{x}|)$, one has $|L_{\Delta_i}(\bar{x};)| = q_{\Delta_i}(|w|)$ (thus, a fortiori $|L_{\Delta_i}(\bar{x};)| \geq q_{\Delta_i}(|w|)$). Therefore, for any input $\bar{x}$, given a configuration code w, $\Delta_i(w) = \hat{\Delta}_i(L_{\Delta_i}(\bar{x};); w)$ where $\hat{\Delta}_i$ and L_{Δ_i} are in $\mathbf{ST}_{\mathsf{P}}$. This means that (reusing the symbol Δ_i) we can consider a function term $\Delta_i(\bar{x}; w) = \hat{\Delta}_i(L_{\Delta_i}(\bar{x};); w)$ in $\mathbf{ST}_{\mathsf{P}}$, which for any input $\bar{x}$ and a given configuration code w returns the next configuration code according to δ_i. Let us define an auxiliary sorted function, called *RUN*. *RUN* is defined, in $\mathbf{ST}_{\mathsf{P}}$, by SR. For a path p and a (initial) configuration code $c(\bar{x};)$, *RUN* simulates the (sequential) computation performed by M along the branch p starting with the configuration code $c(\bar{x};)$.

$$
\begin{aligned}
RUN(\epsilon, \bar{x};) &= c(\bar{x};)\\
RUN(p0, \bar{x};) &= \Delta_0(\bar{x}; RUN(p, \bar{x};))\\
RUN(p1, \bar{x};) &= \Delta_1(\bar{x}; RUN(p, \bar{x};))
\end{aligned}
$$

Let us consider the function f defined by TR[$\boxplus$] in $\mathbf{ST}_{\mathsf{PP}}$:

$$
\begin{aligned}
f(p, \epsilon, \bar{x};) &= RUN(p, \bar{x};)\\
f(p, z0, \bar{x};) &= \boxplus(p, z0, f(p0, z, \bar{x};), f(p1, z, \bar{x};);)\\
f(p, z1, \bar{x};) &= \boxplus(p, z1, f(p0, z, \bar{x};), f(p1, z, \bar{x};);).
\end{aligned}
$$

By construction, one has that $M(\bar{x}) = f(\epsilon, t(\bar{x};), \bar{x};)$. ◀

As a consequence, one can prove the following.

▶ **Proposition 8.** *PP is contained in* $\mathcal{B}(\mathbf{ST}_{PP})$.

Proof. Any boolean function in PP can by definition be computed by a polynomially clocked PTM. It is thus immediate to conclude, from the previous lemma, that PP is contained in $\mathcal{B}(\mathbf{ST}_{\mathsf{PP}})$. ◀

Before moving to the dual result, it is instructive to take a further look at how Lemma 7 is proved. Actually, closure of $\mathbf{ST}_{\mathsf{PP}}$ by TR[⊞] is exploited just once. However, that single use of TR[⊞] takes a non-boolean function as base function. So, although PP is a class of boolean functions, the non-boolean functions of the algebra play a crucial role in the characterization. To show that the boolean functions of the algebra, whose definitions involve TR[⊞] with arbitrary polytime base functions, remain within the complexity class PP is nontrivial, and is essentially what we are going to prove in the next section.

4.2 The Upper Bound

Knowing that $\mathbf{ST}_{\mathsf{P}}$ characterizes FPTIME, it is clear that $\mathcal{B}(\mathbf{ST}_{\mathsf{P}})$ – which corresponds to P, the class of the polytime boolean-functions – is contained in PP. But how about the inclusion between $\mathcal{B}(\mathbf{ST}_{\mathsf{PP}})$ and PP? This is precisely what we are going to prove in this section.

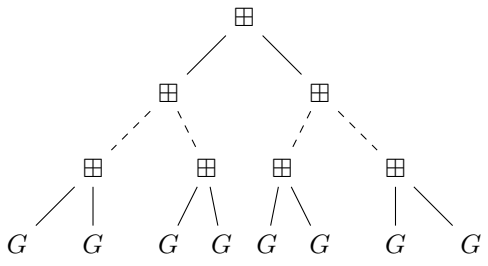
We first of all need the following lemma, which tells us, essentially, that TR[⊞] makes sense from the point of view of the class PP. In this statement only boolean base functions are considered.

▶ **Lemma 9.** *Let G be in P and let F be a function defined as follows:*

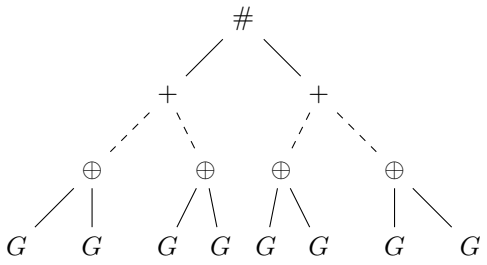
$$\begin{aligned} F(p,\epsilon,\bar{x}) &= G(p,\epsilon,\bar{x}) \\ F(p,z0,\bar{x}) &= \boxplus(p,z0,F(p0,z,\bar{x}),F(p1,z,\bar{x})) \\ F(p,z1,\bar{x}) &= \boxplus(p,z1,F(p0,z,\bar{x}),F(p1,z,\bar{x})). \end{aligned}$$

Then the function F_ϵ defined as $F_\epsilon(z,\bar{x}) = F(\epsilon,z,\bar{x})$, is in PP. Moreover, if G is computable in time t_G (on the sum of the length of its inputs), then $F(\epsilon,z,\bar{x})$ is clocked by $c\cdot|z| +_{\mathbb{N}} t_G(|p| +_{\mathbb{N}} |z| +_{\mathbb{N}} \sum_i |x_i|) +_{\mathbb{N}} 1$, for some constant c.

Proof. In order to compute F_ϵ it is enough to compute something with the following structure



where G is in P and, at each step, the first input (pointer) increases one bit and the second input (recursion input) decreases one bit. At the root of the tree the pointer is ϵ, and at all leaves the recursion input is ϵ. Therefore, what we really have is



Intuitively this is clearly in PP. In order to prove it we consider PTMs with as many tapes as the arity of F (and G). We also assume that each component of the input is placed in one of the tapes (by the order that they show up), i.e. the n-th input is on the n-th tape, and that the machines are initialized with the heads scanning the right most non-empty cell (if there is one).

Let BRANCH be a TM with initial state q_B, which works according to two transition functions – Left and Right. Informally speaking, Left adds the bit "0" at the end of the first input (in the first tape), and deletes the last bit of the second input (in the second tape). The heads move one cell to the right and one cell to the left, respectively. Right proceeds in an analogous way, but adds the bit "1". This can be done in constant time.

G is in P, so there exists a deterministic TM, M_G, which computes the boolean function G, let us say, in time bounded by t_G. We denote by q_G the initial state of M_G, and we assume that the output – 0 or 1 – is written in the last tape.

A PTM for F, M_F, can then be described as follows:

(1) if the head of the second tape scans ϵ, go to (3);
otherwise, go to state q_B and (2);
(2) run BRANCH and (1);
(3) go to state q_G and (4);
(4) run M_G.

We say that final configurations of M_F are accepting configurations if, and only if, the rightmost bit of the last tape is 1.

To determine the computing time of M_F notice that the only possibility of going into a loop is when an instruction calls a previous one, i.e. in (2). Moreover, notice that, inputting $p, z, \bar{x}$ to the machine, (1) is called $|z|$-times, and each loop uses constant time. Therefore, before reaching the instruction (3) the machine performs $c \cdot |z|$ steps, for some constant c. (3) is one step, and (4) uses, at most, t_G steps (notice that the inputs change along the process, but the sum of their lengths remains constant). Thus M_F runs in time bounded by $c \cdot |z| +_{\mathbb{N}} t_G(|p| +_{\mathbb{N}} |z| +_{\mathbb{N}} \sum_i |x_i|) +_{\mathbb{N}} 1$.

M_F accepts $(\epsilon, z, \bar{x})$ if, and only if, more than half of the computations of M_F on $(\epsilon, z, \bar{x})$ end in the ACCEPT final state. Noticing that M_F final configurations are M_G final configurations, we have that M_F accepts $(\epsilon, z, \bar{x})$ if, and only if, more than half of $G(p, \epsilon, \bar{x})$ with $|p| = |z|$ end by 1. Abbreviate $0 \cdots 0$ and $1 \cdots 1$, of length $|z|$, by $0^{|z|}$ and $1^{|z|}$ respectively. Considering the lexicographic order, between $0^{|z|}$ and $1^{|z|}$ we have all $0 - 1$ words with $|z|$ bits. There are $2^{|z|}$ different paths p of length $|z|$. So, M_F accepts $(\epsilon, z, \bar{x})$ if, and only if[2], $\sum_{p=0^{|z|}}^{1^{|z|}}$ last-bit of $G(p, \epsilon, \bar{x}) > \frac{2^{|z|}}{2}$, or equivalently, if and only if $\sum_{p=0^{|z|}}^{1^{|z|}} 2 \cdot$ last-bit of $G(p, \epsilon, \bar{x}) > 2^{|z|}$. This is exactly what TR[⊞] tests: $\oplus$ doubles the last bit of it inputs and add them; $+$ adds the inputs; and # (at the root of the tree) tests whether the global sum (i.e. $\sum_{p=0^{|z|}}^{1^{|z|}} 2 \cdot$ last-bit of $G(p, \epsilon, \bar{x})$) is greater than $2^{|z|}$ – it returns 1 if YES, and 0 otherwise. ◀

[2] In numeric notation.

Let us now show that any boolean function in $\mathbf{ST}_{\mathsf{PP}}$ can be seen as one in the class PP.

▶ **Proposition 10.** *$\mathcal{B}(\mathbf{ST}_{PP})$ is contained in PP.*

Proof. It is enough to show that, for all $f \in \mathbf{ST}_{\mathsf{PP}}$, the function F such that $F(\bar{x}, \bar{y})$ is 1 if $f(\bar{x}; \bar{y})$ ends by 1, and $F(\bar{x}, \bar{y}) = 0$ otherwise, is in PP. We prove this by induction on the definition of the function terms inside $\mathbf{ST}_{\mathsf{PP}}$.

Whenever the scheme TR[⊞] is not involved in the definition of f, one has that $f \in \mathbf{ST}_{\mathsf{P}}$ and therefore the result is immediate. Thus, the relevant cases are the ones where the TR[⊞] is used in the definition of the function term. There are two cases:

- First of all, let f be defined by TR[⊞] with base function $g \in \mathbf{ST}_{\mathsf{P}} \subseteq \mathbf{ST}_{\mathsf{PP}}$. Let $G \in \mathsf{PP}$ be given by induction hypothesis. Consider

$$F(p, z, \bar{x}) = \begin{cases} G(p, \epsilon, \bar{x}) & \text{if } z = \epsilon \\ 0 & \text{if } p, z \neq \epsilon \\ f(\epsilon, z, \bar{x};) & \text{if } p = \epsilon \wedge z \neq \epsilon. \end{cases}$$

  F results from easy case distinctions involving three functions: G and the constant function equal to 0 are in PP. Notice that for $z \neq \epsilon$, $f(\epsilon, z, \bar{x};) = \mathrm{TR}[⊞](g)(\epsilon, z, \bar{x};) = \mathrm{TR}[⊞](\textsf{last-bit of } g)(\epsilon, z, \bar{x};)$, where last-bit of w is $\mathbf{C}(w, 0, 0, 1;)$. g belongs to FPTIME, so last-bit of g is a boolean function in P. Thus, for $z \neq \epsilon$, $f(\epsilon, z, \bar{x};)$ is equal to a function – TR[⊞](last-bit of g)$(\epsilon, z, \bar{x};)$ – which is in PP due to Lemma 9. So, $F \in \mathsf{PP}$. It remains to prove that

$$F(p, z, \bar{x}) = \begin{cases} 1 & \text{if } f(p, z, \bar{x};) \text{ ends by 1} \\ 0 & \text{otherwise.} \end{cases}$$

  Let us further distinguish three sub-cases:

  1. If $z = \epsilon$, then

$$F(p, \epsilon, \bar{x}) = G(p, \epsilon, \bar{x}) = \begin{cases} 1 & \text{if } g(p, \epsilon, \bar{x};) \text{ ends by 1} \\ 0 & \text{otherwise} \end{cases} = \begin{cases} 1 & \text{if } f(p, \epsilon, \bar{x};) \text{ ends by 1} \\ 0 & \text{otherwise.} \end{cases}$$

  2. If $p, z \neq \epsilon$, then $f(p, z, \bar{x})$ is the value returned to the root of a tree as described in Figure 2(b) of Section 3, because the function # is not involved (the pointer only increases along the recursion, therefore if it is not ϵ at the root, it is never ϵ). Thus, as explained in Section 3, the value returned to root of the tree is the binary representation of an even number. Therefore, $f(p, z, \bar{x})$ does not end by the bit 1. Consequently, in this case, $F(p, z, \bar{x}) = 0$.
  3. If $p = \epsilon$ and $z \neq \epsilon$ then, noticing that $f(\epsilon, z, \bar{x};)$ is a single bit, we have that

$$F(\epsilon, z, \bar{x}) = f(\epsilon, z, \bar{x};) = \begin{cases} 1 & \text{if } f(p, \epsilon, \bar{x};) \text{ ends by 1} \\ 0 & \text{otherwise.} \end{cases}$$

  Hence, the function $F \in \mathsf{PP}$ defined above, is 1 if f ends by 1, and it is 0 otherwise.
- If f is defined by SC_P, let us say $f(\bar{x}; \bar{y}) = h(\bar{r}(\bar{x};); \bar{s}(\bar{x}; \bar{y}))$ with $\bar{r}, \bar{s} \in \mathbf{ST}_{\mathsf{P}}$. By induction hypothesis for h, the function H such that $H(\bar{x}, \bar{y})$ is 1 if $h(\bar{x}; \bar{y})$ ends by 1, $H(\bar{x}, \bar{y}) = 0$ otherwise, is in PP. Let M_H be a PTM computing H in polynomial time p_H. $\bar{r}, \bar{s}$ are in $\mathbf{ST}_{\mathsf{P}}$, so let $M_{\bar{r}}$ and $M_{\bar{s}}$ be the correspondent deterministic Turing machines working in polynomial time. One may define the desired machine for F in the obvious way. First

running the deterministic polytime machines $M_{\bar{r}}$ and $M_{\bar{s}}$ in order to produce the input $(\bar{r}(\bar{x};);\bar{s}(\bar{x};\bar{y}))$. Let us say that this is done in time dominated by $p_{\bar{r},\bar{s}}(|\bar{x}|,|\bar{y}|)$. Second, running M_H on this input. The resulting machine is a PTM which computes F and works in time dominated by $p_{\bar{r},\bar{s}}(|\bar{x}|,|\bar{y}|)+_{\mathbb{N}} p_H(|\bar{r}(\bar{x};)|,|\bar{s}(\bar{x};\bar{y})|)$. Evoking now the monotonicity of the polynomials and knowing that the length of $\mathbf{ST}_{\mathsf{P}}$ functions (i.e. FPtime functions) is polynomial bounded – let us say $|\bar{r}(\bar{x};)| \leq q_{\bar{r}}(|\bar{x}|)$ and $|\bar{s}(\bar{x};)| \leq q_{\bar{s}}(|\bar{x}|,|\bar{y}|)$ for some polynomials $q_{\bar{r}}$ and $q_{\bar{s}}$ – we have that the working time of M_F, on the input $\bar{x},\bar{y}$, is bounded by the polynomial $p_{\bar{r},\bar{s}}(|\bar{x}|,|\bar{y}|)+_{\mathbb{N}} p_H(q_{\bar{r}}(|\bar{x}|),q_{\bar{s}}(|\bar{x}|,|\bar{y}|))$.

This finishes the proof. ◀

4.3 Wrapping Up

From Proposition 8 and Proposition 10 one concludes that

▶ **Theorem 11.** $\mathbf{ST}_{PP}$ *characterizes* PP *(i.e.* $PP = \mathcal{B}(\mathbf{ST}_{PP})$*).*

This establishes a purely recursion theoretic characterization of the probabilistic class of complexity PP by adding a specific form of tree-recursion to FPTIME functions. One should notice that the step function of the tree-recursion makes use of the pointer p and the recursion variable z. The same happens in the characterization of FPSPACE given in [13], but it contrasts with the similar characterization of NP given in [14]. For NP the tree-recursion scheme is actually a tree-iteration scheme, i.e. pointers and recursion variable are used only at the bottom (and not all the way along the tree). There the pointers and recursion variable are not taken as inputs of the step function. It is, for instance, not known which class one obtains by restricting the tree-recursion of [13] to tree-iteration.

Whenever working with the tree-recursion scheme or with restricted forms of it, as it is the case here and in the papers mention above, one adopts $\mathbb{W}$ (i.e. $0-1$ words) as base algebra, instead of $\mathbb{N}$. The pointers and the tree-recursion scheme have a natural formulation over $\mathbb{W}$, but the present work can be rewritten in numeric notation. Actually, the seminal paper of Bellantoni and Cook for FPTIME [2], which uses recursion on notation only, is in numeric notation.

A similar characterization of PP can be obtained working in a non-sorted context, by use of explicit bounds on the recursion on notation scheme. Such formulation of PP is based on Cobham's characterization of FPTIME [4] and uses the TR[⊞] scheme neglecting the ";". There is no need of imposing explicit bounds to the non-sorted version of TR[⊞], because due to the fixed step function the scheme is implicitly polynomial bounded.

5 Related Work

The recursion-theoretic approach to implicit computational complexity has proved to be remarkably robust, with many different classes between logarithmic and polynomial space characterized by various forms of recursion schemes [2, 9, 10, 11, 12, 13, 14]. Tree recursion with pointers, in particular, is known to be capable of capturing classes larger than P. But no probabilistic class of complexity was known to admit a recursion-theoretic characterization, so our result is certainly novel. With the given syntactic approach, one cannot expect to capture truly semantic classes like BPP and ZPP, but it may serve as a recursion-theoretic substratum for characterizations of these classes. In addition, it may provide as basis for proof-theoretic characterizations as provided for FPTIME and related classes by Strahm [15] or for the polynomial hierarchy of functions in [8].

Other logical approaches to computational complexity have faced the challenge of characterizing probabilistic classes. As an example, a study of randomization and derandomization in descriptive complexity is due do Eickmeyer and Grohe [5], who were also able to characterize BPP in fixed-point logic with counting. The relationships between theories of bounded arithmetic and probabilistic complexity classes have been studied by Jerábek [7]. The present work complements these approaches by exploiting the framework of safe/tiered recursion [2, 9].

References

1 José L. Balcázar, Josep Díaz, and Joaquim Gabarró. *Structural Complexity I.* Springer Publishing Company, Incorporated, 2nd edition, 2012.

2 Stephen Bellantoni and Stephen A. Cook. A new recursion-theoretic characterization of the polytime functions. *Comput. Complex.*, 2:97–110, 1992.

3 Stephen Bellantoni and Isabel Oitavem. Separating *NC* along the delta axis. *Theor. Comput. Sci.*, 318(1-2):57–78, 2004.

4 Alan Cobham. The intrinsic computational difficulty of functions. In *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.

5 Kord Eickmeyer and Martin Grohe. Randomisation and derandomisation in descriptive complexity theory. *Log. Methods Comput. Sci.*, 7(3), 2011.

6 John Gill. Computational complexity of probabilistic turing machines. *SIAM J. Comput.*, 6(4):675–695, 1977.

7 Emil Jerábek. Approximate counting in bounded arithmetic. *J. Symb. Log.*, 72(3):959–993, 2007.

8 Reinhard Kahle and Isabel Oitavem. Applicative theories for the polynomial hierarchy of time and its levels. *Annals of Pure and Applied Logic*, 164(6):663–675, 2013.

9 Daniel Leivant. Ramified recurrence and computational complexity I: word recurrence and polytime. In P. Clote and J. Remmel, editors, *Feasible Mathematics II*, pages 320–343. Birkhäuser, 1995.

10 Daniel Leivant and Jean-Yves Marion. Ramified recurrence and computational complexity II: substitution and poly-space. In *Computer Science Logic, 8th International Workshop, CSL '94, Kazimierz, Poland, September 25-30, 1994, Selected Papers*, volume 933 of *LNCS*, pages 486–500. Springer, 1994.

11 Peter Møller Neergaard. A functional language for logarithmic space. In Wei-Ngan Chin, editor, *Programming Languages and Systems: Second Asian Symposium, APLAS 2004, Taipei, Taiwan, November 4–6, 2004. Proceedings*, volume 3302 of *LNCS*, pages 311–326. Springer, 2004.

12 Isabel Oitavem. Characterizing *NC* with tier 0 pointers. *Math. Log. Q.*, 50(1):9–17, 2004.

13 Isabel Oitavem. Characterizing PSPACE with pointers. *Math. Log. Q.*, 54(3):323–329, 2008.

14 Isabel Oitavem. A recursion-theoretic approach to NP. *Ann. Pure Appl. Log.*, 162(8):661–666, 2011.

15 Thomas Strahm. Theories with self-application and computational complexity. *Information and Computation*, 185:263–297, 2003.