Baire Categories on Small Complexity Classes and Meager-Comeager Laws

Philippe Moser*

Abstract

We introduce two resource-bounded Baire category notions on small complexity classes such as P, QUASIPOLY, SUBEXP and PSPACE and on probabilistic classes such as BPP, which differ on how the corresponding finite extension strategies are computed. We give an alternative characterization of small sets via resource-bounded Banach-Mazur games. As an application of the first notion, we show that for almost every language A (i.e. all except a meager class) computable in subexponential time, $\mathsf{P}^A = \mathsf{BPP}^A$. We also show that almost all languages in PSPACE do not have small nonuniform complexity.

We then switch to the second Baire category notion (called locally-computable), and show that the class SPARSE is meager in P. We show that in contrast to the resource-bounded measure case, meager-comeager laws can be obtained for many standard complexity classes, relative to locally-computable Baire category on BPP and PSPACE.

Another topic where locally-computable Baire categories differ from resource-bounded measure is regarding weak-completeness: we show that there is no weak-completeness notion in P based on locally-computable Baire categories, i.e. every P-weakly-complete set is complete for P. We also prove that the class of complete sets for P under Turing-logspace reductions is meager in P, if P is not equal to $\mathsf{DSPACE}(\log n)$, and that the same holds unconditionally for QUASIPOLY.

Finally we observe that locally-computable Baire categories are incomparable with all existing resource-bounded measure notions on small complexity classes, which might explain why those two settings seem to differ so fundamentally.

1 Introduction

Both resource-bounded Baire categories and resource-bounded measure were introduced by Lutz in [Lut90, Lut92] for both complexity classes E and EXP. They provide a means of investigating the sizes of various subsets of E and EXP, and give a notion of small sets, called meager sets. Both resource-bounded measure and resource-bounded Baire category have been successfully used to understand the structure of the exponential time classes E and EXP.

Similarly to resource-bounded measure [Lut90], Lutz's formulation for Baire categories is a general theory which holds for many standard deterministic complexity classes containing E ranging from ESPACE to REC. Unfortunately Lutz's formulation does not work on feasible complexity classes contained in E like P; one reason for this is that the characteristic sequence of a language is exponentially larger than the strings whose membership bits it is coding for. Thus simply reading such a characteristic sequence is above the computational power of P. Moreover Baire categories are defined via functions (called finite extension strategies)

*Address: Email: mosersan@gmail.com

extending characteristic sequences of languages, i.e. of the form $h(w) = wu$; thus computing the image of such a function is also above the computational power of P. Whereas some answers to this problem were proposed for the resource-bounded measure case [AS94, May94, Str97, Mos06b], the question was still left open in the Baire category setting.

In this paper, we propose two Baire category notions on small complexity classes, like P, QUASIPOLY, SUBEXP and PSPACE, which differ solely on how the corresponding finite extension strategies are computed. The idea is that instead of computing the whole image of some finite extension strategy $h(w) = wu$, we only require the extension u to be computable in polynomial time.

Ideally, a measure notion in quantitative complexity should satisfy the *three basic properties*:

1. Every singleton set is small.

2. Enumerable infinite unions of small sets are small.

3. The whole class is not small.

These basic properties meet the essence of Lebesgue measure and ensure that no class is both large and small. We show that both Baire category notions introduced in this paper satisfy the three basic properties.

In the classical setting, Baire categories can be alternatively characterized by Banach-Mazur games (see [Oxt80]), which are infinite two-player games, where each player alternatively extends the string output by the other player. We show that Baire category notions on small complexity classes can also be recharacterized in terms of Banach-Mazur games, similarly to Baire categories on EXP [Lut90].

There is another limitation of Lutz's formulation of Baire category [Lut90] (which also occurs in resource-bounded measure [RS98, Mos06b]): it works well for deterministic classes, but not for probabilistic ones. We remedy this situation by introducing a Baire category notion on the class BPP.

As an application of the first notion, we answer a variant of a question raised in [AS94], by showing that almost all (all except a meager class) languages computable in subexponential time, are hard enough to derandomize BPP, i.e. a polynomial time algorithm can use almost any language $L \in$ SUBEXP to derandomize every probabilistic polynomial time algorithm, even if the probabilistic algorithm also has oracle access to L (whereas in [AS94], the probabilistic algorithm has no access to L). We also investigate the nonuniform complexity of languages in PSPACE, and show that almost all languages in PSPACE do not have small nonuniform complexity. A preliminary version of the first Baire category notion introduced in this paper was published in [Mos03].

Although the first Baire category notion introduced here has interesting applications, the class of languages of subexponential density is not small relative to it. To overcome this, Section 4 introduces a second, stronger, Baire category notion, called locally-computable. The second notion is an adaptation of a previous improvement of Lutz's [Lut90, Lut92] notion by Fenner [Fen95], to the setting of small complexity classes. The idea in [Fen95] is to consider finite extension strategies whose image is locally computable in a given time-bound instead of globally computable, which yields a stronger Baire category notion on the class E. Similar to Lutz's [Lut90, Lut92] notion, the Baire category notion of [Fen95] only holds on deterministic complexity classes above (and including) E. In Section 4 we extend Baire categories from

[Fen95], (called local categories) to small complexity classes like P, QUASIPOLY, SUBEXP, PSPACE and BPP.

Informally speaking, a class is said to be meager if there is a computable finite extension strategy that given the prefix of the characteristic sequence of any language in the class, extends it to a string which is no longer a prefix of the characteristic sequence of the language. In Section 3 the extension of the finite extension strategy is required to be polynomial time computable. For locally computable finite extension strategies, we only require the extension to be bit-wise polynomially computable, similarly to [Fen95]. This means that the output of locally computable finite extension strategies can be of any finite size, which yields a stronger resource-bounded Baire category notion than the one from Section 3: the class of languages with subexponential density is meager, relative to this second Baire category notion.

Next we investigate meager-comeager laws in the Baire category setting. A well studied topic in resource-bounded measure deals with understanding which subclasses of EXP satisfy the zero-one law, i.e. classes that have either measure zero or one in EXP. Zero-one laws were obtained for all three probabilistic classes BPP, RP and ZPP [Mel00, IM03]. These laws tell us that either probabilistic algorithms are in some sense weak, or randomness is intractable. Recently a small-or-large law for SPP was proved in [Hit04]. Although resource-bounded measure notions were introduced on almost all small complexity classes [AS94, May94, Str97, Mos06b], no zero-one law has been obtained for the measure notion on PSPACE, nor the measure on BPP yet, i.e. we have no example of classes which have either measure zero or one in PSPACE (nor in BPP). We show that for local Baire category on PSPACE things are different: every standard class contained in PSPACE is either meager in PSPACE or equal to PSPACE. The same holds for replacing PSPACE with BPP, yielding that either derandomization is possible i.e. BPP = P, or P is small compared to BPP.

Table 1: Classes Satisfying a Small-Large Law

Zero-one Laws in	for Resource-bounded Measure	for Baire Category
E	ZPP, RP, BPP [IM03, Mel00]	ZPP, RP, BPP, NP [Fen95]
SUBEXP	ZPP, RP, BPP [Mos06b]	ZPP, RP, BPP, NP [Section 4.5]
PSPACE	?	P, ZPP, RP, BPP, NP [Section 4.5]
BPP	?	P, ZPP, RP [Section 4.5]

Another area where resource-bounded measure and Baire categories on small complexity classes seem to differ is regarding weak-completeness. A language A in some class C is said to be C-weakly-complete [Lut95] if its lower span, i.e. the class of sets reducible to A, does not have C-measure zero. Lutz showed in [Lut95] the existence of E-weakly-complete sets that are not E-complete. Similarly we can define a categorical weak completeness notion, by calling a set loc-weakly-complete if its lower span is not loc-meager. Whereas it is not known whether strictly P-weakly-complete sets (i.e sets that are P-weakly-complete but not P-complete) exist in the resource-bounded measure setting, we show that strictly P-loc-weakly-complete languages do not exist, i.e. every P-loc-weakly-complete language is also P-complete.

We also prove that the class of complete languages for P under Turing-logspace reductions is loc-meager in P, if P is not equal to DSPACE($\log n$), and that the same holds unconditionally for QUASIPOLY, contrasting with the lack of any such result in the resource-bounded measure setting.

Finally we observe that locally-computable Baire categories on P are incomparable to the resource-bounded measure notions on P from [Mos06b], in the sense that every set which is random for P-computable martingales is meager for local categories on P; and that there are generic sets for local categories on P which have P-measure zero. This shows that the size notion derived from both P-measure and local categories on P differ fundamentally (the same holds for $\mathsf{QUASIPOLY}$, SUBEXP, ..., E), which might explain why most of our applications are not known to hold in the setting of resource-bounded measure on small complexity classes.

2 Preliminaries

For complexity classes we use the notation from [BDG95, BDG90, Pap94]. To give a general theory on small complexity classes, we use the following formalism. A family of time bounds is a set of functions $\Delta \subset \{t : \mathbb{N} \to \mathbb{N}, t \text{ is computable }\}$. The time bounds we shall consider are $\text{poly} = \cup_{k\in\mathbb{N}} O(n^k)$, $\text{quasipoly} = \cup_{k\in\mathbb{N}} O(n^{\log^k n})$, $\text{quasipolylin} = \cup_{k\in\mathbb{N}} O(n^{k \log n})$ and $\text{subexp}_\epsilon = \cup_{\delta<\epsilon} O(2^{n^\delta})$ (where $0 < \epsilon < 1$), and let

$$\text{SMALL} = \{\text{poly}, \text{quasipoly}, \text{quasipolylin}, \text{subexp}_\epsilon\}.$$

For a family of time bounds $\Delta \in \text{SMALL}$, we define its corresponding (small) complexity classes $\text{T}(\Delta) = \cup_{t\in\Delta}\mathsf{DTIME}(t)$, $\text{S}(\Delta) = \cup_{t\in\Delta}\mathsf{DSPACE}(t)$ and $\text{BP}(\Delta) = \cup_{t\in\Delta}\mathsf{BPTIME}(t)$. The (small) complexity classes we shall be interested in are $\mathsf{P} = \text{T}(\text{poly})$, $\mathsf{QUASIPOLY} = \text{T}(\text{quasipoly})$, $\mathsf{QUASIPOLY}_{\mathsf{lin}} = \text{T}(\text{quasipolylin})$, $\mathsf{E}_\epsilon = \text{T}(\text{subexp}_\epsilon)$ (where $0 < \epsilon < 1$), $\mathsf{SUBEXP} = \cap_{\epsilon>0}\mathsf{E}_\epsilon$, $\mathsf{PSPACE} = \text{S}(\text{poly})$, and $\mathsf{BPP} = \text{BP}(\text{poly})$.

Regarding $\mathsf{QUASIPOLY}_{\mathsf{lin}}$ and $\mathsf{QUASIPOLY}$, notice that whereas it is easy to show that the canonical complete language (i.e. whose strings are of the form a padding followed by an index i and a string x, such that the ith machine accepts string x in at most t steps, where t depends on the size of the padding) is complete for $\mathsf{QUASIPOLY}_{\mathsf{lin}}$, it is not clear whether it is complete for $\mathsf{QUASIPOLY}$.

Let us fix some notations for strings and languages. A *string* is an element of $\{0,1\}^n$ for some integer n. For a string x, its length is denoted by $|x|$. $s_0, s_1, s_2 \ldots$ denotes the standard enumeration of the strings in $\{0,1\}^*$ ordered by length and then lexicographically, where $s_0 = \lambda$ denotes the empty string. Note that $|w| = 2^{O(|s_{|w|}|)}$. For a string s_i define its position by $\text{pos}(s_i) = i$. If x, y are strings, we write $x \leq y$ if $|x| < |y|$ or $|x| = |y|$ and x precedes y in lexicographical order. A *sequence* is an element of $\{0,1\}^\infty$. If w is a string or a sequence and $1 \leq i \leq |w|$ then $w[i]$ and $w[s_i]$ denote the ith bit of w. Similarly $w[i \ldots j]$ and $w[s_i \ldots s_j]$ denote the ith through jth bits. $\text{dom}(w)$ denotes the domain of w, where w is viewed as a partial function.

For two string x, y, the concatenation of x and y is denoted xy. If x is a string and y is a string or a sequence extending x i.e. $y = xu$, where u is a string or a sequence, we write $x \sqsubseteq y$. We write $x \sqsubset y$ if $x \sqsubseteq y$ and $x \neq y$.

A *language* is a set of strings. A *class* is a set of languages. The cardinality of a language L is denoted $|L|$. Let n be any integer. The set of strings of size n of language L is denoted $L^{=n}$. Similarly $L^{\leq n}$ denotes the set of strings in L of size at most n. We identify a language L with its characteristic function χ_L, where χ_L is the sequence such that $\chi_L[i] = 1$ iff $s_i \in L$. Thus a language can be seen as a sequence in $\{0,1\}^\infty$.

Let A be any language. The lower span (resp. upper span) of A, denoted $A^{\geq^p_m}$ (resp. $A^{\leq^p_m}$) is the set of languages B such that $B \leq^p_m A$ (resp. $A \leq^p_m B$).

For $a, b \in \mathbb{N}$ let $a \dot{-} b$ denote $\max(a - b, 0)$.

2.1 Pseudorandom Generators

The hardness of a generator is the size of the smallest circuit which can distinguish the output of the generator from truly random bits. More precisely,

Definition 2.1 *Let A be any language. The hardness $H^A(G)$ of a random generator $G : \{0,1\}^m \longrightarrow \{0,1\}^n$, is defined as the minimal s such that there exists an n-input circuit C with oracle gates to A, of size at most s, for which:*

$$\left| \Pr_{x \in \{0,1\}^m}[C(G(x)) = 1] - \Pr_{y \in \{0,1\}^n}[C(y) = 1] \right| \geq \frac{1}{s}.$$

The A-oracle circuit complexity of a Boolean function f, denoted $\mathsf{SIZE}^A(f)$ is defined as the size of the smallest circuit with oracle to A computing f. It was discovered in [KvM99] that the construction of pseudorandom generator from high circuit complexity Boolean functions does relativize.

Theorem 2.1 (Klivans-Melkebeek) *Let A be any language. There is a polynomial-time computable function $F : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$, with the following properties. For every $\epsilon > 0$, there exists $a, b \in \mathbb{N}$ such that for any $n \in \mathbb{N}$*

$$F : \{0,1\}^{n^a} \times \{0,1\}^{b \log n} \to \{0,1\}^n,$$

and if r is the truth table of a $(a \log n)$-variables Boolean function of A-oracle circuit complexity at least $n^{\epsilon a}$, then the function $G_r(s) = F(r, s)$ is a generator, mapping $\{0,1\}^{b \log n}$ into $\{0,1\}^n$, which has hardness $H^{\mathrm{A}}(G_r) > n$.

2.2 Finite Extension Strategies

Whereas resource-bounded measure is defined via martingales, resource-bounded Baire categories require finite extension strategies. Here is a definition.

Definition 2.2 *A function $h : \{0,1\}^* \to \{0,1\}^*$ is a finite extension strategy, or a constructor, if for every string $\tau \in \{0,1\}^*$, $\tau \sqsubseteq h(\tau)$.*

For simplicity we use the word *strategy* for finite extension strategy. We say a strategy h avoids some language A (or language A avoids strategy h) if for every string $\tau \in \{0,1\}^*$ we have

$$h(\tau) \not\sqsubseteq \chi_A \,.$$

We say a strategy h meets some language A if h does not avoid A.

We often consider indexed strategies. An indexed strategy is a function

$$h : \mathbb{N} \times \{0,1\}^* \to \{0,1\}^*$$

such that $h_i := h(i, \cdot)$ is a strategy for every $i \in \mathbb{N}$. Let h be an indexed strategy. Consider the following function ext. Let $\sigma \in \{0,1\}^*$ and $i, k \in \mathbb{N}$ and let w be the unique string such that $h_i(\sigma) = \sigma w$. Define

$$\mathrm{ext}(h_i(\sigma), k) = \begin{cases} w[k] & \text{if } 1 \leq k \leq |w| \\ \perp & \text{otherwise.} \end{cases}$$

and

$$\mathrm{ext}(h_i(\sigma)) = w \ .$$

3 Baire Category on Small Complexity Classes

For the rest of this paper, let $\Delta \in \text{SMALL}$ be a family of bounds and let $\mathsf{C} = \mathrm{T}(\Delta)$. be the corresponding small time complexity class (for instance $\mathsf{C} = \mathsf{P}$). Note that most results in this paper also hold for small space-bounded classes, i.e. of the form $\mathsf{C}_S = \mathrm{S}(\Delta)$ (for instance $\mathsf{C}_S = \mathsf{PSPACE}$).

To define resource bounded Baire categories on C, we consider strategies computed by Turing machines which have random access to their inputs, i.e. on input τ, the machine can query any bit of τ to its oracle. For such a random access Turing machine M running on input τ, we denote this convention by $M^{\tau}(\cdot)$. Note that random access Turing machines can compute the lengths of their input τ in $O(\log|\tau|)$ steps (by checking $\tau[1], \tau[2], \tau[2^2], \cdots, \tau[2^i]$ until they go outside the string, followed by a binary search). We consider random access Turing machines running in time $t(|s_{|\tau|}|)$ (equivalently $t(\log|\tau|)$ for some $t \in \Delta$. Nevertheless, for the time-bounded case, such machines cannot read their entire input (because time bounds in Δ are less than exponential).

The idea to define a Baire category notion on C, is to consider strategies whose extension is computable in time $t \in \Delta$, instead of requiring the whole output to be computable, which would not be possible in time $t \in \Delta$.

Definition 3.1 *An indexed strategy $h : \mathbb{N} \times \{0,1\}^* \to \{0,1\}^*$ is Δ-computable if there is a random access Turing machine M as before, such that for every $\tau \in \{0,1\}^*$ and every $i \in \mathbb{N}$,*

$$M^{\tau}(i) = \mathrm{ext}(h_i(\tau)) \tag{1}$$

where M runs in time $t(\log|\tau| + |i|)$, for some $t \in \Delta$. For the space bounded case, we also require the output tape to be bounded, i.e. $|\mathrm{ext}(h_i(\tau))| \leq t(\log|\tau| + |i|)$.

We say a class is small if there is an indexed strategy that avoids every language in the class. More precisely,

Definition 3.2 *A class X of languages is C-meager if there exists a Δ-computable indexed strategy h, such that for every $L \in X$ there exists an index i such that h_i avoids L.*

A class is called comeager if its complement is meager.

In order to formalize the second basic property we need to define *enumerable infinite unions* precisely.

Definition 3.3 *$X = \bigcup_{i\in\mathbb{N}} X_i$ is a C-union of C-meager sets, if there exists an indexed Δ-computable strategy*

$$h : \mathbb{N} \times \mathbb{N} \times \{0,1\}^* \to \{0,1\}^*$$

(i.e. for any $i, j \in \mathbb{N}, \tau \in \{0,1\}^*$, $h(\tau, i, j)$ is computable in time $t(\log|\tau| + |i| + |j|)$, for some $t \in \Delta$) such that for every $i \in \mathbb{N}$, $h_{i,\cdot}$ witnesses X_i's meagerness.

Let us prove the three basic properties.

Theorem 3.1 *For any language L in C, the singleton $\{L\}$ is C-meager.*

Proof. Let $L \in \mathsf{C}$ be any language. We describe a Δ-computable constructor h which avoids $\{L\}$. Consider the following Turing machine M computing h. On input string σ, M^{σ} simply outputs $1 - L(s_{|\sigma|+1})$. h is clearly Δ-computable, and h avoids $\{L\}$. □

The proof of the second basic property is easy.

Theorem 3.2 *A* C*-union of* C*-meager sets is* C*-meager.*

Proof. It is easy to see that a Δ-computable strategy

$$h : \mathbb{N} \times \mathbb{N} \times \{0,1\}^* \to \{0,1\}^*$$

can be transformed into a Δ-computable strategy

$$h' : \mathbb{N} \times \{0,1\}^* \to \{0,1\}^*$$

using a pairing function to combine $\mathbb{N} \times \mathbb{N}$ into $\mathbb{N}$. □

Let us prove the third basic property which says that the whole space C is not small. The idea of the proof is given a strategy h, construct a language L that meets it, where L's characteristic sequences is divided into blocks of exponential size, where block i is used to meet h_i.

Theorem 3.3 C *is not* C*-meager.*

Proof. Let h be an indexed Δ-computable constructor and let M be a Turing machine computing h, running in time $t \in \Delta$. We construct a language $L \in \mathsf{C}$ which meets h_i for every i. The idea is to construct a language L with the following characteristic function,

$$\chi_L = \underbrace{0}_{B_0} \underbrace{\text{ext}(h_1(B_0))0\cdots0}_{B_1} \cdots \underbrace{\text{ext}(h_i(B_0B_1\cdots B_{i-1}))0\cdots0}_{B_i} \tag{2}$$

where block B_i corresponds to all strings of size i, and block B_i contains

$$\text{ext}(h_i(B_0B_1\cdots B_{i-1}))$$

followed by a padding with 0's. B_i is large enough to contain $\text{ext}(h_i(B_0B_1\cdots B_{i-1}))$ because M runs in time t, therefore $|\text{ext}(h_i(B_0B_1\cdots B_{i-1}))| \leq t(\log(|B_0B_1\cdots B_{i-1}|)) < 2^i$ because $t \in \Delta \in$ SMALL (for the space-bounded case, it follows from the requirement on the size of the output tape, see Definition 3.1).

Let us construct Turing machine N deciding L, running in time $t' \in \Delta$. On input x, where $n = |x|$,

1. Compute p where x is the pth word of length n.
2. For $i = 1$ to n simulate $M^{B_0B_1\cdots B_{i-1}}(i)$. Answer M's queries with the previously stored binary sequences $\bar{B}_1, \bar{B}_2, \bar{B}_{i-1}$ in the following way. Suppose that during its simulation $M^{B_0B_1\cdots B_{i-1}}(i)$ queries the kth bit of $B_0B_1\cdots B_{i-1}$ to its oracle. To answer this query, compute the position p_k of s_k among words of size $|s_k|$. Look up whether the stored binary sequence $\bar{B}_{l_k}$ contains a p_kth bit b_k. If this is the case answer M's query with b_k, else answer M's query with 0. Finally store the output of $M^{B_0B_1\cdots B_{i-1}}(i)$ under $\bar{B}_i$.
3. If the stored binary sequence $\bar{B}_n$ contains a pth bit then output this bit, else output 0 (x is in the padded zone of B_n).

Let us check that L is in C. The first and third step are clearly computable in time $O(t(n))$. For the second step we have that for each of the n recursive steps there are at most $t(n)$ queries and each simulation of M once the queries are answered takes at most time $t(n)$. Thus L is computable in time t^4, i.e. $L \in \mathsf{C}$. Note that all $\bar{B}_i$'s have size at most $t(n)$, therefore it's no problem to store them. □

Remark 3.1 *Notice that as opposed to existing notion of resource-bounded measure on small classes [AS94, Str97, Mos06b], this Baire category notion does not need dependency sets, i.e. polynomial printable sets corresponding to the bits read in the input (whereas this restriction is needed for locally-computable categories).*

3.1 Resource-Bounded Banach-Mazur Games

We give an alternative characterization of small sets via resource-bounded Banach-Mazur games, similarly to the classical case (see [Oxt80]) and the resource-bounded case [Lut90]. Informally speaking, a Banach-Mazur game, is a game between two strategies f and g, where the game begins with the empty string λ. Then $g \circ f$ is applied successively on λ. Such a game yields a unique infinite string, or equivalently a language, called the result of the play between f and g. We say that g is a winning strategy for class X if it forces the result of the game with any strategy f to be a language not in X. We show that the existence of a winning strategy for X is equivalent to the meagerness of X. This equivalence result is useful in practice, since it is often easier to find a winning strategy rather than a finite extension strategy.

Definition 3.4

1. *A play of a Banach-Mazur game is a pair (f, g) of strategies such that g strictly extends every string, i.e. for every string $\tau \in \{0,1\}^*$, $\tau \sqsubset g(\tau)$.*

2. *The result $R(f, g)$ of the play (f, g) is the unique element of $\{0,1\}^\infty$ that extends $(g \circ f)^i(\lambda)$ for every $i \in \mathbb{N}$.*

For a class of languages X and time bound families Δ_I and Δ_{II}, denote by $G[X, \Delta_I, \Delta_{II}]$ the Banach-Mazur game with distinguished set X, where player i ($i \in \{I, II\}$) must choose a strategy Δ_i-computable. We say player II wins the play (f, g) if $R(f, g) \notin X$, otherwise we say player I wins. We say player II has a winning strategy for the game $G[X, \Delta_I, \Delta_{II}]$, if there exists a Δ_{II}-computable strategy g such that for every Δ_I-computable strategy f, player II wins (f, g). We denote by $\mathbb{N}^\mathbb{N}$ the class of all functions mapping strings to strings.

The following result states that a class is C-meager iff there is a winning strategy for player II. The idea of the proof is to construct a non-indexed strategy from an indexed one where at step i, the smallest index that has not been met is used.

Theorem 3.4 *Let X be any class of languages. The following are equivalent.*

1. *Player II has a winning strategy for $G[X, \mathbb{N}^\mathbb{N}, \Delta]$.*

2. *X is C-meager.*

Proof. Suppose the first statement holds and let g be a Δ-computable wining strategy for player II. Let M be a Turing machine computing g, in time $t \in \Delta$. We define an indexed Δ-computable constructor h. For $k \in \mathbb{N}$ and $\sigma \in \{0,1\}^*$, define

$$h_k(\sigma) := g(\sigma') \quad \text{where } \sigma' = \sigma 0^{k \dot{-} |\sigma|} \ . \tag{3}$$

h is Δ-computable because computing $h_k(\sigma)$ simply requires simulating $M^{\sigma'}$, answering M's queries in $\mathrm{dom}(\sigma') \backslash \mathrm{dom}(\sigma)$ by 0. We show that if language A meets h_k for every $k \in \mathbb{N}$,

then $A \notin X$. This implies that X is C-meager as witnessed by h. To do this we show that for every $\alpha \sqsubset \chi_A$ there is a string β such that,

$$\alpha \sqsubseteq \beta \sqsubseteq g(\beta) \sqsubset \chi_A \ . \tag{4}$$

If this holds, then player I has a winning strategy yielding $R(f,g) = A$ (unless $A \notin X$): for a given α player I extends it to obtain the corresponding β, thus forcing player II to extend to a prefix of χ_A. So let α be any prefix of χ_A, where $|\alpha| = k$. Since A meets h_k, there is a string $\sigma \sqsubset \chi_A$ such that

$$\sigma' \sqsubseteq g(\sigma') = h_k(\sigma) \sqsubset \chi_A \tag{5}$$

where $\sigma' = \sigma 0^{k \dot{-} |\sigma|}$. Since $|\alpha| \leq |\sigma'|$ and α, σ' are prefixes of χ_A, we have $\alpha \sqsubseteq \sigma'$. Define β to be σ'.

For the other direction, let X be C-meager as witnessed by h, i.e. for every $A \in X$ there exists $i \in \mathbb{N}$ such that h_i avoids A. Let N be a Turing machine computing h, running in time $t \in \Delta$. We define a Δ-computable constructor g inducing a winning strategy for player II in the game $G[X, \mathbb{N}^{\mathbb{N}}, \Delta]$. We show that for any strategy f, $R(f,g)$ meets h_i for every $i \in \mathbb{N}$, which implies $R(f,g) \notin X$. Here is a description of a Turing machine M computing g. For a string σ, M^σ does the following.

1. Compute $n_0 = \min\{t : t \leq \log|\sigma|, \text{ and } (\forall \tau \sqsubseteq \sigma \text{ such that } |\tau| \leq \log|\sigma|) \ \ h_t(\tau) \not\sqsubseteq \sigma\}$.

2. If no such n_0 exists output 0.

3. If n_0 exists (h_{n_0} is the next strategy to be met), simulate $N^\sigma(n_0)$, denote N's answer by ω. Output ω.

g is computable in time $O(n^2 t(n))$, i.e. Δ-computable. We show that $R(f,g)$ meets every h_i for any strategy f. Suppose for a contradiction that this is not the case, i.e. there is a strategy f such that $R(f,g)$ does not meet h. Let m be the smallest index such that $R(f,g)$ does not meet h_m. Since $R(f,g)$ meets h_{m-1} there is a string τ such that

$$h_{m-1}(\tau) \sqsubset R(f,g) \ .$$

Since g strictly extends strings at every round, after at most $2^{O(|\tau|)}$ rounds, f outputs a string σ long enough to enable step one (of M's description) to find out that

$$h_{m-1}(\tau) \sqsubseteq \sigma \sqsubset R(f,g)$$

thus incrementing $m-1$ to m. At this round we have

$$g(\sigma) = \sigma \operatorname{ext}(h_m(\sigma))$$

i.e.

$$h_m \sqsubset R(f,g)$$

which is a contradiction. $\square$

3.2 Application: Derandomization of BPP

It was shown in [AS94] that for every $\epsilon > 0$, for all languages $A \in \mathsf{E}_\epsilon$ except a measure zero class BPP is contained in P^A, thus leaving open the question whether the result also holds if the probabilistic algorithms also have access to A. We answer this question affirmatively in the Baire category setting, i.e. we show that for every $\epsilon > 0$, all languages $A \in \mathsf{E}_\epsilon$ except a meager class satisfy $\mathsf{P}^A = \mathsf{BPP}^A$. The idea of the proof is to construct a strategy that given a initial segment of a language, extends it by the characteristic sequence of a language with high circuit complexity, by diagonalizing over all small circuits. The language can then be plugged into a pseudorandom generator to obtain full derandomization.

Theorem 3.5 *For every $\epsilon > 0$, the set of languages A such that $\mathsf{P}^A \neq \mathsf{BPP}^A$ is E_ϵ -meager.*

Proof. Let $\epsilon > 0$ be small. Let $0 < \delta < \epsilon$ and let $b > 0$ be some integer to be determined later. Consider the following strategy h, computed by the following Turing machine M. On input σ, where $n = \log |\sigma|$, M does the following. At start $Z = \emptyset$, and $i = 1$. M computes z_i in the following way. First if

$$s_{|\sigma|+i} \neq 0^{2^{b|u|}} u$$

for any string u where $|u| = \log(n^{1/b})$, then let $z_i = 0$, output z_i, and compute z_{i+1}; else denote by u_i the corresponding string u. Construct the set T_i of all truth tables of $|u_i|$-input Boolean circuits C with oracle gates for σ of size less than $2^{\delta|u_i|}$, such that

$$C(u_j) = z_j \quad \text{for every} \quad (u_j, z_j) \in Z \ .$$

Compute

$$M_i = \text{Majority}_{C \in T_i}[C(u_i)]$$

and let $z_i = 1 - M_i$. Add (u_i, z_i) to Z. Output z_i, and compute z_{i+1}, unless $u_i = 1^{\log(n^{1/b})}$ (i.e. u_i is the last string of size $\log(n^{1/b})$), in which case M stops.

There are $2^{n^{4\delta/b}}$ circuits to simulate, and simulating such a circuit takes time $O(n^{4\delta/b})$, by answering its queries to σ with the input σ. Finally computing the majority M_i takes time $2^{O(n^{4\delta/b})}$. Thus the total running time is less than $2^{n^{2c\delta/b}}$ for some constant c, which is less than $2^{n^{\epsilon'}}$ (with $\epsilon' < \epsilon$) for an appropriate choice of b.

Let A be any language and consider

$$F(A) := \{u | 0^{2^{b|u|}} u \in A\}.$$

It is clear that $F(A) \in \mathsf{E}^A$. Consider H_δ^A the set of languages L such that every n-input circuits with oracle gates for A of size less than $2^{\delta n}$ fails to compute L. We have

$$F(A) \in H_\delta^A \quad \text{implies} \quad \mathsf{P}^A = \mathsf{BPP}^A$$

by Theorem 2.1.

We show that h avoids every language A such that

$$F(A) \notin H_\delta^A \, .$$

So let A be any such language. There is a n-inputs circuit family $\{C_n\}_{n>0}$, with oracle gates for A, of size less than $2^{\delta n}$ computing $F(A)$. We have

$$C(u_i) = 1 \text{ iff } 0^{2^{b|u_i|}} u_i \in A \quad \text{for every string } u_i \text{ such that } (u_i, z_i) \in Z. \tag{6}$$

(for simplicity we omit C's index). Consider the set D_n of all circuits with $\log(n^{1/b})$-inputs of size at most $n^{\delta/b}$ with oracles gates for A satisfying Equation 6. We have $|D_n| \leq 2^{n^{4\delta/b}}$. By construction, every z_i such that $(u_i, z_i) \in Z$ reduces the cardinal of D_n by a factor 2. Since there are $n^{1/b}$ z_i's such that $(u_i, z_i) \in Z$, we have

$$D_n \leq 2^{n^{4\delta/b}} \cdot 2^{-n^{1/b}}$$

which is less than one because δ is small (smaller than $1/4$). i.e. $D_n = \emptyset$. Therefore $h(\sigma) \not\sqsubset \chi_A$. □

3.3 Almost every Language in PSPACE Does Not Have Small Nonuniform Complexity

The following result shows that almost every language in PSPACE does not have small nonuniform complexity.

Theorem 3.6 *Let For every $c > 0$, $\mathsf{SIZE}(n^c)$ is PSPACE-meager.*

Proof. Let $c > 0$. Consider the following strategy which on input σ with $n = \log|\sigma|$, outputs a string u of size $2n^{c+1}$ defined as follows. Let u_i denote the ith bit of u, and let z_{u_i} be the string whose membership bit corresponds to u_i, in the characteristic sequence starting with σu. For $n \leq t$ denote by $S(n, t, \sigma, u_1 \cdots u_l)$ the set of n-inputs Boolean circuits of size t such that $C(z_{u_j}) = u_j$ for every $1 \leq j \leq l$. Let

$$u_i = 1 - \text{Majority}\{C(z_{u_i}) : C \in S(|z_{u_i}|, |z_{u_i}|^c, \sigma, u_1 \cdots u_{i-1})\}.$$

It is well known [Pap94] that

$$|S(n, t, \sigma, u_1 \cdots u_t)| \leq 2^{t \log t}$$

thus h is computable in $\mathsf{DSPACE}(n^{c+2})$ by constructing all corresponding circuits. h avoids $\mathsf{SIZE}(n^c)$, because there are less than $2^{n^{c+1}}$ such circuits, and each bit u_i of the extension u diagonalizes against at least half such circuits. □

The proof of Theorem 3.3 shows that the class of all languages with subexponential density is not C-meager. In the next section, we improve the power of Δ-computable strategies by considering locally computable strategies, which can avoid the class of languages of subexponential density.

4 Locally Computable Categories on Small Complexity Classes

In order to allow random access Turing machines to compute the length of their inputs τ, without querying their oracles (due to the query set requirement that follows), we also provide them with $s_{|\tau|}$. For such a Turing machine M running on input τ, we denote this convention by $M^{\tau}(s_{|\tau|})$.

Similarly to [Fen95], we shall consider strategies whose extensions are bit-wise computable in time $t \in \Delta$. Such strategies are very strong since the extension can be of any finite size, as long as it is locally computable. As it is usually the case with most notions of measure defined in small complexity classes [AS94, Str97, Mos06b], to be able to show that the whole class is not small, the power of the Turing machines computing the strategies needs to be

reduced, by requiring that all queries made to the input are contained in a t-printable set ($t \in \Delta$), called the query set (a set S is t-printable if there is a Turing machine M which on input 1^n outputs all strings in $S^{=n}$ in time $t(n)$).

Definition 4.1 *An indexed strategy $h : \mathbb{N} \times \{0,1\}^* \to \{0,1\}^*$ is called Δ-loc-computable if there exists a random access Turing machine M as before such that for every $\tau \in \{0,1\}^*$ and every $i, k \in \mathbb{N}$,*

$$M^\tau(s_{|\tau|}, i, k) = \mathrm{ext}(h_i(\tau), k)$$

where M runs in time $t(\log|\tau| + |i| + |k|)$ for some $t \in \Delta$, and there is a t-printable query set G such that for every $n, i, k \in \mathbb{N}$ and for every $i', k' \in \mathbb{N}$ such that $i' \leq i$ and $k' \leq k$ and for every input $\sigma \in \{0,1\}^*$ such that $\log|\sigma| \leq n$, $M^\sigma(s_{|\sigma|}, i', k')$ queries σ only on bits that are in $G(n, i, k)$, where $G(n, i, k)$ is printable in time $t(n + |i| + |k|)$.

A class of languages is called meager if there is an indexed strategy that avoids every language in the class.

Definition 4.2 *A class X of languages is C-loc-meager if there exists a Δ-loc-computable indexed strategy h, such that for every $L \in X$ there exists $i \in \mathbb{N}$, such that h_i avoids L.*

The definition of *enumerable infinite unions* is similar to Definition 3.3.

4.1 The Three Basic Properties

Let us check that all three basic properties hold for locally-computable Baire categories.

Theorem 4.1 *For any language L in C, the singleton $\{L\}$ is C-loc-meager.*

Proof. Follows from Theorem 3.1. □

The following result states that enumerable infinite unions of small sets are small.

Theorem 4.2 *A C loc-union of C loc-meager sets is C loc-meager.*

Proof. Similar to Theorem 3.2. □

Let us prove the third basic property. The proof idea is similar to that of Theorem 3.3, i.e. given a strategy h we construct a language L that meets h, where L's characteristic sequence is divided into blocks where on the ith block, h_i is met. The difference is that now we do not know the size of the extension computed by h_i, therefore we first show the existence of a function that bounds the size of h_i, which we use to determine the size of block i.

Theorem 4.3 *C is not C loc-meager.*

Proof. We need the following technical Lemma, that bounds the size of the extensions efficiently.

Lemma 4.1 *Let h be a Δ-loc-computable indexed strategy. Then there exists a function $f : \mathbb{N} \to \mathbb{N}$ such that,*

1. *For every $\sigma \in \{0,1\}^*$ such that $|\sigma| \leq \sum_{j=0}^{i-1} f(j)$ we have $|\mathrm{ext}(h_i(\sigma))| \leq f(i)$,*
2. *$f(0) = 1$ and $f(i) \geq 2^i$ for every $i \in \mathbb{N}$,*

and there exists a deterministic Turing machine which on input i, computes $f(i)$ within $O(\log(f(i)))$ steps.

Proof. Let h be any Δ-loc-computable indexed strategy and let N be a Turing machine witnessing this fact. We construct a deterministic Turing machine M for f. At each step of the computation, M increments a counter R. On input i, M computes f recursively as follows: $f(0) = 1$. For $i > 0$ compute

$$B = \sum_{j=0}^{i-1} f(j) \ .$$

For every string σ of size at most B, simulate $N^{\sigma}(s_{\sigma}, i, k)$ for $k = 1, 2, \ldots$ until N outputs $\perp$, store the corresponding k under k_{σ}. Compute

$$K := \max_{|\sigma| \leq B} k_{\sigma}.$$

Stop incrementing the counter R, compute 2^{R+K+i}, and output this value.

For the running time of M on input i, observe that the last two steps (once the counter R is stopped) take time $O(R+K+i)$. Thus the total running time of M is at most $O(R+K+i)$ which is less than $O(\log(f(i)))$. This ends the proof of the Lemma.

Let us prove the Theorem. Let h be a Δ-loc-computable indexed constructor and let M be a Turing machine computing h with query set G_M. Let f be as in Lemma 4.1 and let Q be a Turing machine computing f.

We construct a language $L \in \mathsf{P}$ which meets h_i for every i. The idea is to construct a language L with the following characteristic function,

$$\chi_L = \underbrace{0}_{B_0} \underbrace{\mathrm{ext}(h_1(B_0))0\cdots0}_{B_1} \underbrace{\mathrm{ext}(h_2(B_0B_1))0\cdots0}_{B_2} \cdots \underbrace{\mathrm{ext}(h_i(B_0B_1\cdots B_{i-1}))0\cdots0}_{B_i}$$

where block B_i has size $f(i)$ and contains $\mathrm{ext}(h_i(B_0B_1\cdots B_{i-1}))$ followed by a padding with 0's. B_i is large enough to contain $\mathrm{ext}(h_i(B_0B_1\cdots B_{i-1}))$ by definition of f.

Let us construct a Turing machine N deciding L. On input x, where $n = |x|$,

1. Compute $\mathrm{pos}(x)$.

2. Compute the index $i(x)$, where the membership bit of x is in zone $B_{i(x)}$, with the formula

$$i(x) = \max_{j \geq 0} [\sum_{t=0}^{j-1} f(t) < \mathrm{pos}(x) + 1]$$

in the following way. At the beginning $S = 1$, then for $t = 1, 2, \ldots$ compute $f(t)$ by simulating Q for $|x|^2$ steps, and add the result to S, until either Q doesn't halt, or $S \geq \mathrm{pos}(x) + 1$. Let t_0 denote the first t for which this happens, and let

$$i(x) = t_0 - 1 \ .$$

3. Compute the position of x in $B_{i(x)}$ with

$$\mathrm{rpos}(x) = \mathrm{pos}(x) - F(i(x)-1)$$

where

$$F(j) = \sum_{t=0}^{j} f(t) \ .$$

4. Compute the membership bit of x, where

$$\mathrm{bit}(x) = \mathrm{ext}(h_{i(x)}(B_0 B_1 \cdots B_{i(x)-1}), \mathrm{rpos}(x)) \ .$$

If $\mathrm{bit}(x) = \perp$, then output 0 (x is in the padded zone of $B_{i(x)}$), otherwise output $\mathrm{bit}(x)$.

Let us check that L is in C. The first step is clearly computable in time polynomial in n. For the second step notice that if $f(t) < \mathrm{pos}(x)$, Q must halt within $O(\log(\mathrm{pos}(x)))$ steps, which is less than $|x|^2$ since $\mathrm{pos}(x) = 2^{O(|x|)}$. Thus the second step computes $i(x)$ correctly. Moreover since f increases at least exponentially, only a polynomial number of terms need to be summed in the second and third step. Since f is at least exponentially increasing, the sums in step two and three can be done in polynomial time. Finally the last step requires simulating

$$M^{B_0 B_1 \cdots B_{i(x)-1}}(s_{F(i(x)-1)}, i(x), \mathrm{rpos}(x)) \ .$$

By the hypothesis on h, M's queries are all in

$$G_M(|s_{F(i(x)-1)}|, i(x), \mathrm{rpos}(x))$$

which is contained in

$$G_M(|s_{F(i(x)-1)}|, i(x), \mathrm{pos}(x))$$

which has size $t(|x|)$ for some $t \in \Delta$. For such a query q, i.e. suppose M queries the qth bit of its input, simply run step one to four above with x replaced by q. By definition of G_M at most $t(|x|)$ recursive steps need to be performed. □

4.2 Resource-bounded Banach-Mazur Games

Similarly to Section 3.1 we give an alternative characterization of small sets via resource-bounded Banach-Mazur games. The proof is an extension of a similar proof in [Fen95].

Theorem 4.4 *Let X be any class of languages. The following are equivalent.*

1. *Player II has a winning strategy for $G[X, \mathbb{N}^{\mathbb{N}}, \Delta\text{-loc}]$.*
2. *X is C loc-meager.*

Proof. We need the following technical Lemma, that gives an efficient bound on the size of the extensions.

Lemma 4.2 *Let h be a Δ-loc-computable indexed constructor. Then there exists a function $f : \mathbb{N} \to \mathbb{N}$ such that for every $m \in \mathbb{N}$, $t \leq m$ and for every string τ of size at most m, $|h_t(\tau)| \leq f(m)$, and there exists a deterministic Turing machine which on input m, computes $f(m)$ within $O(f(m))$ steps.*

Let us prove the lemma. Let h be a Δ-loc-computable strategy, and let N be a Turing machine witnessing this fact. We construct a deterministic Turing machine computing f. At each step of the computation, M increments a counter R. On input $m \in \mathbb{N}$, compute

$$K = \max_{|\tau| \leq m, t \leq m} \{|h_t(\tau)|\}$$

by simulating N on all appropriate strings. Stop incrementing the counter R, compute $K + R$ and output the result. Thus M's total running time is less than $O(R + K)$ which is in $O(f(m))$. This ends the proof of the Lemma. □

For the proof of the Theorem, suppose the first statement holds and let g be a wining Δ-loc-computable strategy for player II. Let M be a Turing machine computing g. We define an indexed Δ-loc-computable constructor h by constructing a machine N for h; let $i \in \mathbb{N}$ and $\sigma \in \{0,1\}^*$,

$$h_i(\sigma) := g(\sigma') \quad \text{where } \sigma' = \sigma 0^{i \dot{-} |\sigma|}.$$

h is Δ-loc-computable because computing $h_i(\sigma)$ simply requires to simulate $M^{\sigma'}(s_{|\sigma'|})$ answering M's queries in $\mathrm{dom}(\sigma') \backslash \mathrm{dom}(\sigma)$ by 0. Thus N's query set satisfies

$$G_N(|s_{|\sigma|}|, i, k) \subseteq G_M(|s_{|\sigma'|}|, k)$$

which has size $t(\log |\sigma| + |i| + |k|)$ for some $t \in \Delta$, because $|\sigma'| \leq |\sigma| + i$.

We show that if language A meets h_k for every $k \in \mathbb{N}$, then $A \notin X$. This implies that X is C loc-meager as witnessed by h. To do this we show that for every $\alpha \sqsubset \chi_A$ there is a string β such that,

$$\alpha \sqsubseteq \beta \sqsubseteq g(\beta) \sqsubset \chi_A.$$

If this holds, then player I has a winning strategy yielding $R(f, g) = A$: for a given α player I extends it to obtain the corresponding β, thus forcing player II to extend to a prefix of χ_A. So let α be any prefix of χ_A, where $k = |\alpha|$. Since A meets h_k, there is a string $\sigma \sqsubset \chi_A$ such that

$$\sigma' \sqsubseteq g(\sigma') = h_k(\sigma) \sqsubset \chi_A$$

where $\sigma' = \sigma k \dot{-} |\sigma|$. Since $|\alpha| \leq |\sigma'|$ and α, σ' are prefixes of χ_A, we have $\alpha \sqsubseteq \sigma'$. Define β to be σ'.

For the other direction, let X be C loc-meager as witnessed by h, i.e. for every $A \in X$ there exists $i \in \mathbb{N}$ such that h_i avoids A. Let N be a Turing machine computing h. Let $f : \mathbb{N} \to \mathbb{N}$ be as in Lemma 4.2, and let Q be a deterministic Turing machine computing f. We define a Δ-loc-computable constructor g inducing a winning strategy for player II in the game $G[X, \mathbb{N}^{\mathbb{N}}, \Delta\text{-loc}]$. We show that for any strategy s, $R(s, g)$ meets h_i for every $i \in \mathbb{N}$, which implies $R(s, g) \notin X$. Here is a description of a Turing machine M computing g. For a string σ with $n = \log |\sigma|$, $M^{\sigma}(s_{|\sigma|}, k)$ does the following.

1. Compute
$$B = \max_{m \geq 1} [f(m) \leq n]$$
in the following way. For $t = 1, 2, \ldots$ compute $f(t)$ by simulating Q for n^2 steps, and denote the result by b_t, until either Q doesn't halt, or $b_t > n$. Let t_0 denote the first t for which this happens, then define $B = b_{t_0 - 1}$.

2. Compute $n_0 = \min\{t : t \leq B, \text{ and } (\forall \tau \sqsubseteq \sigma \text{ such that } |\tau| \leq B) \;\; h_t(\tau) \not\sqsubseteq \sigma\}$.

3. If no such n_0 exists output 0 if $k = 1$, and output $\perp$ if $k > 1$.

4. If n_0 exists, then if $k = 1$ output 0, otherwise simulate $N^{\sigma 0}(s_{|\sigma|+1}, n_0, k-1)$ answering N's queries in $\mathrm{dom}(\sigma 0)\backslash\mathrm{dom}(\sigma)$ with 0, and output the result of the simulation.

Let us check that g is Δ-loc-computable. For the first step, we have that whenever $f(m) \leq n$, Q halts within $O(n)$ steps. Since Q is simulated n^2 steps, B is computed correctly, in polynomial time. For the second step, the B^3 simulations of N can be done in time $u \in \Delta$. Moreover every $h_t(\tau)$ computed during the second step has size at most B, thus only the first n bits of the input σ need to be read. This together with the fourth step guarantees that the query set for M is given by

$$G_M(|s_{|\sigma|}|, k) = \{1, 2, \ldots n\} \cup G_N(|s_{|\sigma|+1}|, n, k-1)$$

which has size $u'(\log|\sigma|)$, for some $u' \in \Delta$.

We show that $R(s, g)$ meets every h_i for any strategy s. Indeed suppose this is not the case, i.e. there is a strategy s such that $R(s, g)$ does not meet h. Let n_0 be the smallest index such that $R(s, g)$ does not meet h_{n_0}. Since $R(s, g)$ meets h_{n_0-1} there is a string τ such that

$$h_{n_0-1}(\tau) \sqsubset R(s, g) \ .$$

Since g strictly extends strings at every round, after a certain number of rounds, s outputs a string σ long enough to enable step two (of M's description) to find out that

$$h_{n_0-1}(\tau) \sqsubseteq \sigma$$

thus incrementing $n_0 - 1$ to n_0. At this round we have

$$g(\sigma) = \sigma 0 \, \mathrm{ext}(h_{n_0}(\sigma 0))$$

i.e.

$$h_{n_0} \sqsubset R(s, g)$$

which is a contradiction. □

4.3 Local Categories on BPP

In this section we introduce local categories on the probabilistic class BPP. To this end we need the following probabilistic strategies.

Definition 4.3 *An indexed strategy $h : \mathbb{N} \times \{0,1\}^* \to \{0,1\}^*$ is called* BPP*-loc-computable if there is a probabilistic random access Turing machine (as defined in Definition 3.1) M such that for every $\tau \in \{0,1\}^*$ and every $i, k, n \in \mathbb{N}$,*

$$\Pr[M^\tau(s_{|\tau|}, i, k, n) = \mathrm{ext}(h_i(\tau), k)] \geq 1 - 2^{-n}$$

where the probability is taken over the internal coin tosses of M, M runs in time polynomial in $\log|\tau| + |i| + |k| + n$, and there is a poly printable query set G such that for every $m, i, k \in \mathbb{N}$ and for every $i', k' \in \mathbb{N}$ (such that $i' \leq i$ and $k' \leq k$), and for every input $\sigma \in \{0,1\}^*$ (such that $\log|\sigma| \leq m$), $M^\sigma(s_{|\sigma|}, i, k, n)$ queries σ only on bits that are in $G(m, i, k)$; where $G(m, i, k)$ is printable in time polynomial in $m + |i| + |k|$.

Remark 4.1 *By using standard Chernoff bound arguments it is easy to show that Definition 4.3 is robust, i.e. the error probability can range from* $\frac{1}{2}+\frac{1}{p(n)}$ *to* $1-2^{-q(n)}$ *for any polynomials* p, q*, without enlarging or reducing the class of strategies defined this way.*

Similarly to the deterministic case, a class X is called meager if there is a single probabilistic strategy that avoids X.

Definition 4.4 *A class of languages* X *is* BPP*-loc-meager if there exists a* BPP*-loc-computable indexed strategy* h*, such that for every* $L \in X$ *there exists* $i \in \mathbb{N}$*, such that* h_i *avoids* L*.*

The definition of enumerable infinite unions is similar to Definition 3.3.

Let us prove that all three basic properties hold for locally-computable Baire categories on BPP.

Theorem 4.5 *For any language* L *in* BPP*,* $\{L\}$ *is* BPP*-loc-meager.*

Proof. The proof is similar to Theorem 4.1 except that the constructor h is computed with error probability smaller than 2^{-n}. □

The second basic property is easy to show.

Theorem 4.6 *A* BPP*-loc-union of* BPP*-loc-meager sets is* BPP*-loc-meager.*

Proof. Similar to Theorem 3.2. □

Let us prove the third basic property.

Theorem 4.7 BPP *is not* BPP*-loc-meager.*

Proof.

The proof is similar to Theorem 4.3 except for the second step of N's computation, where every simulation of M is performed with error probability smaller than 2^{-n}. Since there are n distinct simulations of M, the total error probability is smaller than $n2^{-n}$, which ensures that L is in BPP. □

4.4 SPARSE is Meager in P

It was shown in Section 3 that the class of languages with subexponential density is not meager for the first category notion in this paper (the language in the proof of Theorem 3.3 has subexponential density). Here we prove that local computable strategies are stronger than the strategies of Section 3, by showing that the class SPARSE (and also the class of languages of subexponential density) is P-loc-meager. The idea of the proof is to extend any prefix of a language with enough ones to make sure it is not sparse.

Theorem 4.8 SPARSE *is* P*-loc-meager.*

Proof. Let L be any sparse language. Then there exists a polynomial p such that

$$|L \cap \{0,1\}^n| \leq p(n) \text{ for every } n \geq 1.$$

Consider the following strategy h, which on input $\sigma \in \{0,1\}^*$ pads σ with $|\sigma|$ 1's. Since L is sparse, h avoids L. We construct a random access Turing machine M for h; on input $\sigma \in \{0,1\}^*$ and $j \in \mathbb{N}$, $M^\sigma(s_{|\sigma|}, j)$ outputs 1 if $1 \leq j \leq |\sigma|$ and $\perp$ otherwise. Since M doesn't query its oracle, h is P-loc-computable which ends the proof. □

4.5 Meager-Comeager Laws

The following meager-comeager laws in PSPACE and BPP contrast with the resource-bounded measure case, where many of those all or nothing laws are not known to hold.

Theorem 4.9 *Let $X \in \{\mathsf{ZPP}, \mathsf{RP}, \mathsf{BPP}, \mathsf{NP}\}$. Then either X is C-loc-meager or $X = \mathsf{C}$.*

Proof. We need the following lemma, whose proof is an extension of a similar result in [Fen95].

Lemma 4.3 *Let X be a Σ_2^0 class, such that there exists a language A in C, such that for every finite variant A' of A, $A' \notin X$. Then X is C loc-meager.*

Let us show the lemma. The idea of the proof is to extend any prefix of a language according to language A until until we know we have avoided X.

By hypothesis there exists a polynomial oracle Turing machine M, such that

$$X = \{L | \exists x \forall y : M^L(x, y) = 0\} \ .$$

Consider the following C loc-computable strategy g where $\mathrm{ext}(g(\sigma), k))$, with $n = \log |\sigma|$, is computed as follows.

1. Simulate $M^L(x, y)$ for every $x < \log n$ and $y < \log k$, where
$$\chi_L = \sigma A(s_{|\sigma|+1}) A(s_{|\sigma|+2}) \cdots$$
2. If for every $x < \log n$ there exists $y < \log k$ such that $M^L(x, y) \neq 0$ output $\perp$, else output $A(s_{|\sigma|+k})$.

Let us show that g is a strategy. Suppose for a contradiction that g extends σ infinitely. Then the result is a finite variant of A, hence not in C. Therefore there exists $k \in \mathbb{N}$, such that

$$(\forall x < |s_{|\sigma|}|)(\exists y < \log k) M^L(x, y) \neq 0$$

where $L = g(\sigma)$. Hence g should not have extended σ more than k bits, which is a contradiction.

g is Δ-loc-computable since there are $\log n \cdot \log k$ simulations to perform and since the queries to A can be computed in t steps ($t \in \Delta$). The query set $G_g(n, k)$ has size $t(n + |k|)$ for some $t \in \Delta$, therefore g is Δ-loc-computable.

Let us show that g avoids X. Denote by L the result of the game between g and some strategy f, where player II plays according to g. Let z be any string. On the first turn for player II where the state of the game is of length at least $2^{2^{z+1}}$, player II extends ensuring that

$$(\forall x < z + 1)(\exists y < \log k) M^L(x, y) \neq 0 \ .$$

Thus

$$M^L(z, y) \neq 0$$

which implies $L \notin X$.

This ends the proof of the lemma. We have the following consequences.

Corollary 4.1 *Let X be a Σ_2^0 class closed under finite variants. Then X is C-loc-meager iff $\mathsf{C} \not\subseteq X$.*

Which ends the proof of the theorem.

Similarly meager-comeager laws in BPP can be proved.

Theorem 4.10 *Let $X \in \{\mathsf{P}, \mathsf{ZPP}, \mathsf{RP}\}$. Then either X is* BPP*-loc-meager or $X = \mathsf{BPP}$.*

Proof. It is easy to check that Lemma 4.3 also holds in BPP. □

4.6 Weak Completeness

The concept of weak completeness was introduced in [Lut95]. A set A is called C-weakly-complete if its lower span (the class of sets reducible to A) does not have C-measure zero. Lutz showed in [Lut95] the existence of EXP-weakly-complete sets that are not EXP-complete. Similarly we can define a categorical weak completeness notion, by calling a set A C-loc-weakly-complete if its lower span is not C-loc-meager. We show that there is no P-loc-weakly-complete incomplete language, i.e. P-loc-weakly-completeness is equivalent to P-completeness.

Theorem 4.11 P*-loc-weakly-completeness is equivalent to* P*-completeness, under Turing logspace reductions.*

Proof. Let $A \in \mathsf{P}$ be any language. It is easy to check that the lower span $A^{\geq_T^{\log}}$ is a Σ_2^0 class and is closed under finite variants. Thus by Corollary 4.1 we have $A^{\geq_T^{\log}}$ is not P-loc-meager iff A is $\leq_T^{\log}$-hard for P. □

Another consequence of Corollary 4.1 is the meagerness of the class of complete sets for P, under the assumption P is not equal to $\mathsf{DSPACE}(\log n)$.

Theorem 4.12 *If* P *is not equal to* $\mathsf{DSPACE}(\log n)$*, then the class of* $\leq_T^{\log}$-P*-complete sets is* P*-loc-meager.*

Proof. Let A be a $\leq_T^{\log}$-P-complete set. Consider $A^{\leq_T^{\log}}$ the upper span of A. It is easy to check that $A^{\leq_T^{\log}}$ verifies the hypothesis of Corollary 4.1. By our assumption, A cannot reduce to a set in $\mathsf{DSPACE}(\log n)$, so $\mathsf{P} \not\subseteq A^{\leq_T^{\log}}$, hence $A^{\leq_T^{\log}}$ is P-loc-meager. Since every $\leq_T^{\log}$-P-complete language is in $A^{\leq_T^{\log}}$, this ends the proof. □

Note that the same result holds unconditionally for locally-computable categories on $\mathsf{QUASIPOLY}_{\mathsf{lin}}$.

Theorem 4.13 *The class of* $\leq_T^{\log}$-$\mathsf{QUASIPOLY}_{\mathsf{lin}}$*-complete sets is* $\mathsf{QUASIPOLY}_{\mathsf{lin}}$*-loc-meager.*

Proof. The proof is similar to Theorem 4.12.

4.7 Measure vs Baire Categories

Although Lemma 4.3 shows that locally computable strategies are extremely strong, the following easy observation shows that the size notion yielded from resource-bounded measure is incomparable with the one derived from Baire category. This might explain why many locally computable Baire category results on small complexity classes (meager-comeager laws in PSPACE and BPP, equivalence between P-loc-weak-completeness and P-completeness, conditional smallness of the class of P-complete languages) are not known to hold in the resource-bounded measure setting on small complexity classes.

We use the measure notion on P of [Mos06b]. We shall give a brief description of it. A martingale is a function $d : \{0,1\}^* \to \mathbb{R}_+$ such that, for every $w \in \{0,1\}^*$,

$$2d(w) = d(w0) + d(w1).$$

A martingale d succeeds on language L if

$$\limsup_{n\to\infty}(L[1\ldots n]) = \infty.$$

Informally speaking, a set has P-measure zero if there exists a martingale d computable in polynomial time that succeeds on every language in the class. For full details we refer the reader to [Mos06b]. A language R is P-random if every polynomial time martingale does not succeed on R. A language G is P-loc-generic if every P-loc-computable strategy does not avoid G.

The following result shows that P-measure and P-loc-categories are incomparable. The idea of the proof is that for any P-random language it is impossible that the language contains no strings of size n for infinitely many lengths n; such a language can then easily be avoided by a locally-computable strategy. The other direction is due to the existence of P-loc-generic languages containing significantly more zeroes that ones in the limit, which makes it possible for a martingale to succeed on them.

Theorem 4.14

1. *Every* P*-random set is* P*-loc-meager.*
2. *There exist* P*-loc-generic sets which have* P*- measure zero.*

Proof. Let R be P-random, then

$$\forall^\infty n : L_n^{=n} \neq \emptyset$$

(where $L_n^{=n}$ denotes the n first strings of size n) otherwise consider the following P-computable martingale d which divides its initial capital into shares $c_n = 1/n^2$, and uses capital c_n to bet on strings of size n, by betting all the capital that the first n strings of size n have membership bit 0. Whenever this bet is correct for strings of size n, d wins $2^n/n^2$. Since

$$\exists^\infty n : L_n^{=n} \neq \emptyset$$

d's capital grows unbounded on R, which contradicts R's P-randomness. Consider the following P-loc-computable strategy h. By hypothesis there exists a constant k such that

$$\forall n > k : L_n^{=n} \neq \emptyset\ .$$

Therefore strategy h defined by

$$\mathrm{ext}(h(\sigma)) = 0^{2|\sigma|+2^{2k}}$$

avoids R, i.e. $\{R\}$ is P-loc-meager.

For the second part consider the following language G. Let M_i be an (non-effective) enumeration of all Turing machines computing P-loc-computable strategies. Consider the following characteristic sequence of G .

$$\chi_G = \underbrace{1}_{B_0}\ \underbrace{\mathrm{ext}(h_1(B_0))}_{B_1}\ \underbrace{0\cdots0}_{B_2}\ \underbrace{\mathrm{ext}(h_2(B_0B_1B_2))}_{B_3}\ \underbrace{0\cdots0}_{B_4}\cdots$$

where block B_{2i} contains $5 \cdot |B_0 B_1 \cdots B_{2i-1}|$ 0's. By construction G is P-loc-generic because it meets every P-loc-computable strategy. Consider the same P-computable martingale d as above. By construction of G, the zones padded with 0's are large enough to guarantee that

$$\exists^{\infty} n : G_n^{=n} = \emptyset \ .$$

Therefore d's capital grows unbounded on G which ends the proof. □

5 Conclusion

We have introduced two Baire category notions on small deterministic and probabilistic classes, and given applications of both notions in derandomization, circuit complexity, meager-comeager laws and weak-completeness, some of which are not known to hold with respect to measure in small complexity classes. We then observed that categories and measure on small classes are incomparable, which might explain these differences between the two settings.

Acknowledgments
We thank the anonymous referees for their useful comments.

References

[AS94] E. Allender and M. Strauss. Measure on small complexity classes, with application for BPP. *Proc. of the 35th Ann. IEEE Symp. on Found. of Comp. Sci.*, pages 807–818, 1994.

[BDG90] J. L. Balcazar, J. Diaz, and J. Gabarro. *Structural Complexity II.* EATCS Monographs on Theoretical Computer Science Volume 22, Springer Verlag, 1990.

[BDG95] J. L. Balcazar, J. Diaz, and J. Gabarro. *Structural Complexity I.* EATCS Monographs on Theoretical Computer Science Volume 11, Springer Verlag, 1995.

[Fen95] S. A. Fenner. Resource-bounded Baire category : a stronger approach. *Proceedings of the Tenth Annual IEEE Conference on Structure in Complexity Theory*, pages 182–192, 1995.

[Hit04] John M. Hitchcock. The size of SPP. *Theoretical Computer Science*, 320:495–503, 2004.

[IM03] R. Impagliazzo and P. Moser. A zero-one law for RP. *Proceedings of the 18th Conference on Computational Complexity*, pages 48–52, 2003.

[KvM99] A. Klivans and D. van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial hierarchy collapses. *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, pages 659–667, 1999.

[Lut90] J.H. Lutz. Category and measure in complexity classes. *SIAM Journal on Computing*, 19:1100–1131, 1990.

[Lut92] J.H. Lutz. Almost everywhere high nonuniform complexity. *Journal of Computer and System Science*, 44:220–258, 1992.

[Lut95] J.H. Lutz. Weakly hard problems. *SIAM Journal on Computing*, 24:1170–1189, 1995.

[May94] Elvira Mayordomo. Measuring in PSPACE. *Proceedings of the 7th International Meeting of Young Computer Scientists (IMYCS'92). Gordon-Breach Topics in Computer Science 6*, 136:93–100, 1994.

[Mel00] D. Melkebeek. The zero-one law holds for BPP. *Theoretical Computer Science*, 244(1-2):283–288, 2000.

[Mos03] P. Moser. Baire's categories on small complexity classes. *14th Int. Symp. Fundamentals of Computation Theory*, pages 333–342, 2003.

[Mos06a] P. Moser. Lp computable functions and Fourier series. *submitted*, 2006.

[Mos06b] P. Moser. Martingales family and dimension in P. *Logical Approaches to Computational Barriers, Second Conference on Computability in Europe, CiE 2006, Swansea, UK*, pages 388–397, 2006.

[Oxt80] J. C. Oxtoby. *Computational complexity.* Springer-Verlag, Berlin, second edition, 1980.

[Pap94] C. Papadimitriou. *Computational complexity.* Addisson-Wesley, 1994.

[RS98] K. Regan and D. Sivakumar. Probabilistic martingales and BPTIME classes. *In Proc. 13th Annual IEEE Conference on Computational Complexity*, pages 186–200, 1998.

[Str97] M. Strauss. Measure on P- strength of the notion. *Inform. and Comp.*, 136:1:1–23, 1997.