Circuit Expressions of Low Kolmogorov Complexity

José L. Balcázar  
Department of Software (LSI)  
Universitat Politècnica de Catalunya  
Pau Gargallo 5, E-08028 Barcelona, Spain  
balqui@lsi.upc.es

Harry Buhrman  
CWI  
Kruislaan 413, P.O. Box 94079  
1090 GB Amsterdam, The Netherlands  
buhrman@cwi.nl

Montserrat Hermo  
Department of Software (LSI)  
Universidad del País Vasco, P.O. Box 649  
E-20080 San Sebastián, Spain  
jiphehum@si.ehu.es

January 5, 1999

Abstract

We study circuit expressions of logarithmic and poly-logarithmic polynomial-time Kolmogorov complexity, focusing on their complexity-theoretic characterizations and learnability properties. They provide a nontrivial circuit-like characterization for a natural nonuniform complexity class that lacked it up to now. We show that circuit expressions of this kind can be learned with membership queries in polynomial time if and only if every NE-predicate is E-solvable. Thus they are learnable given that the learner is allowed the extra use of an oracle in NP. The precise way of accessing the oracle is shown to be optimal under relativization. We present a precise characterization of the subclass defined by Kolmogorov-easy circuit expressions that can be constructed from membership queries in polynomial time, with some consequences for the structure of reduction and equivalence classes of tally sets of very low density.

*Preliminary, sometimes weaker versions of the results in this paper were announced at EuroColt'95. Partially supported by the E.U. through the ESPRIT Long Term Research Project 20244 (ALCOM-IT) and through the HCM Network CHRX-CT93-0415 (COLORET); by the Spanish DGICYT through project PB95-0787 (KOALA); and by Acciones Integradas Hispano-Alemanas HA-119-B and AL-201-B.
1 Introduction

One of the well-developed subareas of Complexity Theory studies the resources spent by algorithms that change the representation of a formal language in different ways. Under the name “Coding Complexity” (see [7]), it is possible to provide a unified conceptual framework for a number of recent advances in the understanding of such processes. One of the most studied tools in that area is the nonuniform Boolean circuit model of computation. For instance, consider the problem of constructing a Boolean circuit, of a given size, that correctly represents at a given length a formal language that can be only accessed through queries about the membership of individual strings. Upper and lower bounds are known for problems of this sort, and they relate to some issues of Computational Learning Theory (see again [7]).

In such a complexity-theoretic setting, circuit expressions play essentially the same role as circuits, without the limitation that the length of the input be always the same. Their construction is similar to that of regular expressions, with the difference that each boolean circuit is considered as a circuit expression, and thus can be combined with other expressions by the usual regular expression operators. They were introduced by Watanabe and Gavalda [29] as a simple variant of boolean circuits that allows for easier technical argumentation, in the above-mentioned context of relationships between Computational Learning and Structural Complexity. Actually, circuit expressions can be substituted for circuits in nearly all the relevant results in the literature of Computational Complexity. This paper presents some of the few cases in which the difference becomes noticeable. Specifically, these are the cases in which strong conditions on the Kolmogorov complexity of the circuit expressions are imposed.

Indeed, there are cases in which assumptions like low Kolmogorov complexity allow for learnability properties that do not hold, or are not known or not expected to hold, without them [21]. Thus, focusing on Kolmogorov-easy circuit expressions, we report here on a study of their complexity-theoretic and learnability properties, as a contribution to the study of Coding Complexity, and along the lines hinted at in [29].

The class of concepts that are represented by such easy circuit expressions is precisely identified here as the nonuniform class called Full-P/log, introduced in [19] (under a different name). We must mention that there are technical facts that prevented, up to now, a circuit-like characterization of this nonuniform class; specifically, the fact that circuits correspond to inputs of fixed length. (See [16] for a circuit characterization of the related class P/log.)

The advantage of Full-P/log lies in its closure under polynomial-time reductions. A complexity-theoretic analysis of this class has been done in [15] (see also [8]), and as we will see throughout this paper, this research provides a framework that helps to investigate efficient learnability of Kolmogorov-easy circuit expressions via queries.

With a rather easy argument, based on the definition of Kolmogorov-easy circuit expressions itself, this representation class can be learned efficiently with equivalence queries without counterexamples; this is explained below. However, when only membership queries are allowed, the learnability problem turns out to be more difficult. We prove that in this case the efficient learnability problem is equivalent to the (unlikely) complexity-theoretic hypothesis that accepting computations of NE machines can be computed in E.
The main results obtained are organized as follows. In Section 3, Kolmogorov-easy circuit expressions are characterized in terms of the nonuniform class Full-P/log. From the structural properties of Full-P/log, some consequences about the learnability via membership queries are found. In Section 4, it is shown that for every concept having easy circuit expressions, these can be identified efficiently with membership queries in the presence of an NP oracle. This turns out to correspond with the fact that Full-P/log is in the first level of the extended low hierarchy [1, 6, 12, 22].

Therefore, under the hypothesis P=NP, Kolmogorov-easy circuit expressions can be efficiently learned with membership queries. Our next result improves on this, weakening the complexity-theoretic assumption to achieve the converse implication. In addition we show that, under a natural notion of relativization, the learning algorithm used to produce these circuit expressions is optimal.

In Section 5, we study the natural subclass for which the circuit expressions can be constructed using the most inexpensive queries: membership without additional oracles. This subclass is again characterized using structural results about Full-P/log. The fact that advice strings for concepts in the nonuniform class can be encoded into special tally sets is the key point for this characterization.

The extension of the last result to various Kolmogorov complexity bounds is also discussed in the last section. Many of the technical properties on which our proofs are based also hold for other bounds, in particular for the polylog case.

2 Preliminaries

An alphabet \( \Sigma \) is any non-empty, finite set. We use here the alphabets \( \{0, 1\} \) and \( \{0\} \). Given any alphabet \( \Sigma \), a finite string (or word) over \( \Sigma \) is a finite sequence of symbols from \( \Sigma \). We denote them by lower case Latin letters, such as \( x, y, z, \ldots \). The \( i \)-th symbol (or bit) of \( x \) is denoted by \( x_i \). Of course, this makes sense when \( i \leq |x| \), the length of \( x \). The notation \( x_{i:j} \) denotes the substring of \( x \) from the \( i \)-th bit on up to and including the \( j \)-th bit.

Any set (finite or infinite) of strings is called a language. The language of all possible finite strings over \( \Sigma \) is denoted by \( \Sigma^* \). Given two words \( x, y \), the concatenation of \( x \) with \( y \) is denoted \( xy \). A prefix of a word \( y \) is any word \( z \) such that for some word \( w, y = zw \), i.e. any chain of symbols which form the beginning of \( y \). The notation \( z \sqsubseteq y \) denotes the fact that \( z \) is a prefix of \( y \).

Sets and, in particular, languages, are denoted by upper case Latin letters. Given a finite set \( A \), we indicate the cardinality of \( A \) by \( |A| \). The set of all strings \( x \) in \( A \) whose length is less than or equal to \( n \) is denoted by \( A^\leq n \). When we refer to the strings in \( A \) of length exactly \( n \), we use the notation \( A^n \). The complement of a language \( A \subseteq \Sigma^* \) is denoted by \( \overline{A} \); when \( \Sigma \) is known, we omit it, leaving just \( \overline{A} \). Given any \( \Sigma \) with at least two symbols, say 0 and 1, the join or marked union of two languages \( A \) and \( B \) over \( \Sigma \) is:

\[
A \oplus B = \{ x0 \mid x \in A \} \cup \{ x1 \mid x \in B \}
\]

**Definition 1** A language is *tally* if and only if it is a subset of \( \{0\}^* \).

3
We will need to encode several words into one in such a way that both computing the encoding, and recovering the coded words can be easily done. Thus we use a pairing function \( \langle \cdot \rangle : \Sigma^* \times \Sigma^* \to \Sigma^* \): given \( x \) and \( y \), the word \( \langle x, y \rangle \) is obtained by duplicating each bit of \( x \), appending to this the word \( y \), inserting a 01 in between. Assuming that the lengths of \( x \) and \( y \) are \( n \) and \( m \) respectively, the length of the pairing function applied to \( (x, y) \) is:

\[
|\langle x, y \rangle| = 2n + 2 + m
\]

The computations of \( \langle \cdot \rangle \) and its inverses use a minimum of resources. The pairing function can be applied to tuples as follows: \( \langle x, y, z \rangle = \langle x, \langle y, z \rangle \rangle \) and so on.

**Uniform computation and Kolmogorov complexity**

Now we introduce some models of computation. Our uniform computational model is the multi-tape Turing machine, with a read-only input tape. An oracle Turing machine has an additional write-only oracle tape. Machines can be deterministic (DTM) or nondeterministic (NDTM), and work as acceptors or as transducers.

We work here with classes of languages recognized by oracle Turing machines that work in polynomial time. Assume that the machine \( M \) queries oracle \( A \). The language \( L \) recognized by \( M \) using \( A \) \( (L = L(M, A)) \) is said to be polynomial-time Turing reducible to oracle \( A \). We denote this fact as \( L \in \text{P}(A) \). Equivalently, \( \text{NP}(A) \) is the class of those languages recognized by oracle NDTM’s querying oracle \( A \) in polynomial time. When a family of oracles \( \mathcal{F} \) is used instead of a particular oracle \( A \), \( \text{P}(\mathcal{F}) \) denotes the following class of languages

\[
\text{P}(\mathcal{F}) = \{ L | \exists A \in \mathcal{F} \text{ such that } L \in \text{P}(A) \}
\]

and equivalently \( \text{NP}(\mathcal{F}) \). NP is the corresponding class with empty oracle.

We also use the concept of Turing equivalence in polynomial time. A language \( L \) is polynomial-time Turing equivalent to another language \( A \) when \( L \in \text{P}(A) \) and \( A \in \text{P}(L) \). The class of all polynomial-time Turing-equivalent languages to \( A \) is denoted by \( \text{E}(A) \). Similarly, given a family of oracles \( \mathcal{F} \),

\[
\text{E}(\mathcal{F}) = \{ L | \exists A \in \mathcal{F} \text{ such that } L \in \text{E}(A) \}
\]

An interesting NP-complete set that plays an important role in Structural Complexity is \( \text{SAT} \). This set is defined as follows:

\[
\text{SAT} = \{ \phi | \phi \text{ is a satisfiable quantifier-free boolean formula} \}
\]

When any variable \( x \) in a formula \( \phi \) is substituted by \( i \in \{0, 1\} \), a new formula is obtained, denoted by \( \phi|_{x=i} \). An important property of \( \text{SAT} \) is self-reducibility [23, 25]. Informally, \( \text{SAT} \) is self-reducible because the satisfiability problem of a boolean formula is “reducible” to the satisfiability problem of smaller formulas: if \( x \) occurs in \( \phi \), then \( \phi \) is satisfiable if and only if at least one of \( \phi|_{x=0} \) and \( \phi|_{x=1} \) is satisfiable.

**Definition 2** A set \( A \) is self-reducible if and only if there exists a deterministic polynomial-time oracle machine \( M \), such that the following holds:
1. \( A = L(M, A) \).

2. On each input of length \( n \), \( M \) queries the oracle only about strings of length at most \( n - 1 \).

Hypothesis \( Q \) was introduced by Allender and Watanabe [2]. \( Q \) is a short name for the following statement, which is not known to hold: for each nondeterministic linear exponential time \( (2^{O(n)}) \) machine accepting an input, it is possible to compute, deterministically, in linear exponential time, one accepting computation\(^1\).

Note that trivially \( Q \) implies that \( E \neq NE \); the converse is known to fail under relativization [17] (see [11] for a proof of this based on Kolmogorov complexity). A similar statement at the NP level is well-known to characterize \( P \neq NP \), via the computation of witnesses provided by the self-reducibility of SAT.

It is well-known that there exist Universal Turing machines able to simulate any other Turing machine given to it encoded as part of the input, with reasonable time and space overheads. Fix such a universal machine \( U \). Using \( U \), we introduce the resource-bounded Kolmogorov Complexity [26, 14]. The original idea of Kolmogorov complexity is modified in order to include the running time spent by the universal Turing machine to produce an output. Kolmogorov complexity strings \( K[f(n), g(n)] \) are defined as follows:

**Definition 3**

\[ K[f(n), g(n)] = \{ x \mid \exists y, |y| \leq f(|x|), U(y) = x \text{ in at most } g(|x|) \text{ steps} \} \]

Sometimes we use classes of sets whose words have low resource-bounded Kolmogorov complexity rather than only sets. They are defined in the following way:

**Definition 4**

\[ K[\mathcal{F}, \mathcal{G}] = \{ A \mid \exists f(n) \in \mathcal{F}, g(n) \in \mathcal{G} \text{ such that } \forall x \in A, x \in K[f(n), g(n)] \} \]

The most used classes of resource-bounded Kolmogorov complexity in this paper are \( K[\log, \text{poly}] \) and \( K[\text{polylog}, \text{poly}] \), where \( \text{poly} = \{ f \mid f(n) \in n^{O(1)} \} \), \( \log = \{ f \mid f(n) \in O(\log n) \} \), and \( \text{polylog} = \{ f \mid f(n) \in O((\log n)^c) \} \) for some constant \( c \). From now on, \( \log n \) means \( \log_2 n \).

**Nonuniform computation and circuit expressions**

Nonuniform computation models are those in which different “programs” are used for different input lengths. The most frequently used model is the boolean circuit, in which binary signals are associated to edges of a directed acyclic graph and transformed at each vertex according to a boolean function associated to that vertex. Some of the vertices have fan-in zero and are associated to input bits. The output signal of a specific node is the result of the circuit.

\( ^1 \)The name \( Q \) is not used in the final publication by Allender and Watanabe, but they used it verbally and is now in common use in conversation. It must be mentioned that some people use the same name for the somewhat different hypothesis that it is possible to produce in deterministic polynomial time accepting computations for nondeterministic polynomial-time machines that accept everything.
The nonuniform computational model we use here is based on circuits, and consists of “circuit expressions”. They were introduced in [29]. In this model, not only one circuit can be considered, but also several of them simultaneously, combined in the same way as regular expressions. The formal definition is as follows:

**Definition 5**

1. If $E$ is a circuit, then $E$ is a circuit expression.

2. If $E_1$ and $E_2$ are circuit expressions, then $E_1 + E_2$ is a circuit expression.

3. If $E$ is a circuit expression, then $E^*$ is a circuit expression.

Hence, a circuit expression is a boolean circuit with a single output, or the sum of two circuit expressions, or a star operator applied to a circuit expression. Let $CE$ be the set of all circuit expressions. The language denoted by any $E \in CE$ is defined in the usual way:

**Definition 6**

1. If $E$ is a circuit, then $L(E)$ is the language recognized by $E$.

2. If $E = E_1 + E_2$, then $L(E) = L(E_1) \cup L(E_2)$.

3. If $E = E_1^*$, then $L(E) = L(E_1)^*$.

Denote by $CEX(A, n)$ the set containing all circuit expressions $C$ such that $L(C)$ coincides exactly with $A$ up to size $n$. Note the fact that circuit expressions can be evaluated in polynomial time (see [15] for a detailed proof):

**Theorem 7** [29]

$CEV P = \{ \langle \langle x, E \rangle \mid E \in CE \text{ and } x \in L(E) \} \}$ is in P.

We are interested in circuit expressions that have easy descriptions in the sense of Kolmogorov Complexity.

**Definition 8** A set $A$ has Kolmogorov-easy circuit expressions if and only if for some constant $c$, and for each $n$, there is $E_n \in CEX(A, n)$, of size $|E_n| \leq n^c$, such that $E_n \in K[c \log n, n^c]$.

The standard way to move from uniform to nonuniform complexity classes is through the use of advice words. We focus now on a nonuniform complexity class proposed by Ker-I Ko in [19], and studied later in [10]. We are interested in this class because, as we will see, it characterizes exactly the languages that have Kolmogorov-easy circuit expressions.

**Definition 9** A set $A$ is in Full-P/log if

$$\forall n \exists w_n (|w_n| \leq c \log n) \forall x (|x| \leq n) x \in A \iff \langle x, w_n \rangle \in B$$

where $B \in P$ and $c$ is a constant.
Full-P/log has been well studied in [8] (see also [15] and [9]) where many characterizations were given. We present one of them, based on the following subclass of the tally sets.

**Definition 10** Tally2 = \{L | L \subseteq \{0^k | k \in \mathbb{N}\}\}


The related class P/log is defined in the same way, but the “∀x” quantifier only ranges over the words of length exactly n. Thus, in P/log, advice word \(w_n\) is good for deciding all words of length n, whereas in Full-P/log it must work as well for all previous lengths. If one considers polynomially long advice (the other most usual case), then both definitions are equivalent, but for logarithmic advice they are not [15].

### 3 Characterizing easy circuit expressions

We characterize now the concepts represented by Kolmogorov-easy circuit expressions, thus giving a circuit-like characterization of Full-P/log. The class P/poly has a well-known characterization in terms of polynomial size circuits, and the class P/log in terms of circuits of logarithmic Kolmogorov complexity (see [16]). P/log consists of all sets having polynomial size circuits whose encodings are in K[log,poly].

Since Full-P/log is provably different from P/log, no natural characterization of Full-P/log seems possible in that way. The introduction of circuit expressions allows for the following alternative:

**Theorem 12** A set \(A\) can be decided by a family of Kolmogorov-easy circuit expressions if and only if \(A \in\) Full-P/log.

**Proof:** (\(\Leftarrow\)) Suppose first that \(A \in\) Full-P/log: there exists a sequence of advice words \(w_n\) with \(|w_n| \leq c \log n\) for some constant c, and a set \(B \in P\), such that for all \(n\) it holds:

\[
\forall x (|x| \leq n) (x \in A \iff \langle x, w_n \rangle \in B)
\]

Simulating the deterministic Turing machine that computes the set \(B\) [24] it is possible to construct from \(n\) and \(w_n\) a polynomial-size circuit \(C_j \in K[\log n, n^d]\) (for some constant \(d\)) that recognizes each \(A^{=j}\) for \(j \leq n\). In more detail, each \(C_j\) includes a simulating circuit for the polynomial-time machine that computes the pairing function, having as “hardwired” (i.e. fixed to constants) inputs the advice bits of \(w_n\) (not \(w_j\)) to pair it with its input, and then piped into a simulating circuit for the polynomial-time machine accepting \(B\) at length \(|\{0^j, w_n\}|\). These circuits \(C_j\), for all \(j \leq n\), have polynomial size but simply consist of replications of constant-size sub-circuits that apply the transition table of the machines, so they are easily described by their width and depth, plus \(w_n\), with in total logarithmically many bits. They can be put together into an easy circuit expression for \(A^{\leq n}\), by means of the union operation. It is important to notice here that the logarithmically many constant gates encoding the advice word are *the same* for all the circuits in the circuit expression.
(⇒) Suppose now that \( A \) has Kolmogorov-easy circuit expressions; thus there exists a constant \( c \) such that, for all \( n \), there exists a circuit expression \( E_n \) in such a way that \( E_n \in CEX(A, n) \), \( |E_n| \leq n^c \), and \( E_n \in K[c \log n, n^c] \). As a first approximation, consider the set \( B \in \text{P} \) formed by pairs \( \langle x, s \rangle \) where \( x \) is in the language recognized by the circuit expression produced by the universal machine, \( U \), from \( s \). The running time of \( U \) is only allowed to be an appropriate polynomial.

It is easy to see that it is not enough to take simply the seeds as advice words. If, together with \( x \), we give a seed of roughly the same length, it takes exponential time to find and evaluate an exponentially larger circuit expression; and, if we forbid this possibility, we are not fulfilling the definition of Full-P/log. The advice words to be taken are, instead, concatenations of the seeds for circuit expressions \( E_n \) for \( n \) a double power of 2. This proof technique has been extensively used in the study of Full-P/log and related Kolmogorov complexity classes ([15], [8]).

More precisely, each advice word will contain several seeds, corresponding to some selected lengths, and avoiding to store all of them. We will encode all the seeds for lengths \( n = 2^m \) and skip all the intermediate ones. For all \( n \) the seed for length \( n \) has itself a length less than \( c \log n \). Let \( w_n \) be the seed for length \( n \) with a padding word from \( 1^\ast \), in order to get \( |w_n| = c \log n \). Now let \( z_m \) be the concatenation of all the words \( w_{2^0}, w_{2^1}, \ldots, w_{2^m} \), and denote \( m_n \) the natural number such that \( 2^{m_n-1} < n \leq 2^{m_n} \). Then the advice word for length \( n \) will be \( z_{m_n} \). The length of \( z_{m_n} \) is exactly the following:

\[
|z_{m_n}| = |w_{2^0}| + |w_{2^1}| + \ldots + |w_{2^m}| = c + 2c + \ldots + 2^m c = (2^{m_n+1} - 1)c
\]

By the choice of \( m_n \), the value \((2^{m_n}+1-1)c\) is bounded by \( 4c \log n \), therefore the length of \( z_{m_n} \) is again logarithmic in \( n \).

We can define the set \( B \) as consisting of pairs. For each \( x \), it holds \( \langle x, s \rangle \in B \), where \( s \) is now a concatenation of words, in such a way that the \( m_x \)-th of them, when used as input for \( U \), yields in polynomial (in \( |x| \)) time a circuit expression accepting \( x \). It is immediate that \( B \in \text{P} \) and that \( x \in A \iff \langle x, z_{m_n} \rangle \in B \) for \( |x| \leq n \).

One may find it paradoxical that, on one hand, all circuits are circuit expressions, but on the other hand the circuits of logarithmic Kolmogorov complexity (i.e. P/log) yield a proper super class of the circuit expressions of logarithmic Kolmogorov complexity (i.e. Full-P/log). The point is that, given a fixed set to be described, the condition on the Kolmogorov complexity is weaker on circuits, which must be good for fixed lengths, than on circuit expressions, which must work for all the previous lengths as well.

The first part of the proof of the theorem indicates that in order to produce circuit expressions it is enough to find a way of computing the appropriate advice words. These, together with the length up to which the circuit expression is desired and a constant size program, are enough to fully reconstruct the desired circuit expression. The mechanism that is followed to output the circuit expression will be useful in next sections.

The second part remarks the fact that we can reorganize the seeds for circuit expressions keeping only the seeds to some selected lengths, instead of storing all of them. This mechanism allows us to optimize the information that is needed to produce the circuit expressions. We will appeal to this in later sections, too.
4 Learnability of easy circuit expressions

This section discusses some learnability properties of the circuit expressions of logarithmic polynomial-time Kolmogorov complexity characterized in the previous section. We work with the “learning via queries” model of Computational Learning, which was introduced by Angluin [3]. In this model, there are algorithms (learners) interacting with a “teacher”, trying to learn a concept. The concept is formalized as a set of positive examples for the concept, and each example is in turn just a binary string. Thus each concept is just a language. The interaction with the teacher is formalized by “queries”. The most natural one is the “membership” query, in which the learner presents an example (a binary word) and asks the teacher whether it belongs to the concept set. We also use the “equivalence” query, where a representation of the concept is given to the teacher, who either declares it correct or provides a counterexample. Here we will actually use the restricted equivalence query, in which the teacher simply answers whether it is correct or not. In more generality, queries can be modeled by complexity-theoretic operators in the sense of Ko [18], since they actually capture a sort of relativized computation in which the target set is used as an oracle.

For instance, using circuit expressions as representations, it is easy to see that the restricted equivalence query to target $A$ can be answered by an oracle in NP($A$).

It remains to define when an equivalence query is “correct”. We work with the “bounded learning” model from [28], where it is not necessary to completely identify the concept: a length bound is given initially to the learner, and it must identify the concept up to that bound. The equivalence query is answered positively as soon as the queried representation coincides with the target set up to that length. A bound is given initially as well on the size of the smallest representation that would get a positive answer. Thus, learning a representation class such as circuit expressions (or a subset $D$ thereof) amounts to finding through queries to a target set $A$ a circuit expression correct for $A$ up to a given length, under the hypothesis that there is one of length below a given bound (and in $D$ if specified).

As a warm-up, we consider now learning from restricted equivalence queries. From the fact that only polynomially many descriptions exist for Kolmogorov-easy circuit expressions, we get easily an algorithm based on equivalence queries: simply, reconstruct all potential circuit expressions by cycling over all logarithmically long seeds, and ask each one as an equivalence query.

The interest of pointing this out is that it allows us to explain that one has to be somewhat more precise: a single polynomial time-bounded algorithm is not able to check all the seeds of size $O(\log n)$, unless the constant in the order of magnitude is explicitly known to it. Thus, the precise statement is that there is a polynomial-time algorithm, making equivalence queries, that can learn exactly Kolmogorov-easy circuit expressions whose seeds are logarithmically long for fixed constants. Formally, we introduce the following parameterization:

**Definition 13** $\text{KCE}_c = \{ E \mid E \in \text{CE} \cap \text{K}[c \log n, n^c]\}$

Then, the algorithm sketched above leads to:

**Proposition 14** A set $D$ of circuit expressions is learnable with equivalence queries in polynomial time if and only if, for some fixed constant $c$ and for every $n$, every expression $E \in D$ has an equivalent expression, up to length $n$, in $\text{KCE}_c$. 

9
Consider the algorithm sketched above: it reconstructs and queries all Kolmogorov-
easy circuit expressions from the seeds of length at most $c \log n$. The reconstruction is made
by simulating the universal Turing machine during exactly $n^c$ steps. This algorithm learns
all circuit expressions of $D$ since one of the queried expressions is indeed equivalent to the
target. (If $D$ does not include $\text{KCE}_c$, this algorithm would be considered “improper learning”
in the sense that the output of the algorithm might not belong to $D$.)

Conversely, assume that an algorithm learns arbitrary circuit expressions with restricted
equivalence queries in time $n^d$. Without loss of generality, we can assume that the last query
made by the learner gets the answer YES, being the unique YES during the computation.
This query, that is the learned circuit expression, can be generated from a description of the
algorithm and a number $e$ indicating that the circuit expression is the $e^{th}$ query made
by the algorithm. Therefore, all the circuit expressions found by this algorithm belong to
$\text{KCE}_{d+1}$.

This easy Proposition 14 proves not only that equivalence queries suffice for each $\text{KCE}_c$,
but also, via a now standard transformation [3], that these classes are PAC-learnable.

From now on, we want to avoid the specification of individual constants in our terminology. Abusing notation somewhat, we will simply say that “$\text{KCE}$ is learnable” to mean that
for each constant $c$ there is a polynomial time algorithm that learns $\text{KCE}_c$. The degree of
the polynomial time bound is allowed to depend on $c$. With this convention, the previous
proposition reads “$\text{KCE}$ is learnable from restricted equivalence queries”.

Next we will treat learnability from much less powerful queries: membership queries. We
start with a fact whose proof is rather standard.

**Theorem 15** Kolmogorov-easy circuit expressions for every set $A \in \text{Full-P}/\log$ can be con-
structed in polynomial time, making logarithmically many queries to $\text{NP}(A)$.

**Proof:** Suppose $A \in \text{Full-P}/\log$. Then

$$\forall n \exists w_n \ (|w_n| \leq c \log n) \ \forall x \ (|x| \leq n) \ (x \in A \iff \langle x, w_n \rangle \in B)$$

where $B \in \text{P}$ and $c$ is a constant. All the constructions here will depend on $c$.

For each $n$, we can construct $w_n$ by a prefix-search algorithm asking queries to $GA$ a set
in $\text{NP}(A)$. (See below for the definition of $GA$). Let $y$ be a word such that $|y| \leq c \log n$. We
say that $y$ is “good” for $n$ (in the sense of being a correct advice) if and only if

$$\forall u \ |u| \leq n \ (\langle u, y \rangle \in B \iff u \in A)$$

Let $GA$ be the following oracle set:

$$GA = \{ \langle z, 0^n \rangle \mid |z| \leq c \log n \text{ and } \exists y \ z \subseteq y, \ |y| \leq c \log n, \text{ and } y \text{ is “good” } \}$$

Then $GA$ is in co-$\text{NP}(A)$ via the following co-$\text{NP}$ algorithm. On input $\langle z, 0^n \rangle$ search among
the polynomial many extensions $y$ of $z$ ($|y| \leq c \log(n)$) whether for one of them it holds that
for all $u$ with length $\leq n : \langle u, y \rangle \in B \iff u \in A$.

It is not hard to see that one can obtain the correct advice $w_n$ by doing a prefix-search
with logarithmically many queries to $GA$. Note that given the length $n$ in unary, and a good
advice word \( w_n \) corresponding to \( n \), it is easy to construct the polynomial-size Kolmogorov-
easy circuit expression \( E_n \) that decides each \( A^{j} \) for \( j \leq n \). This process was explained in
the proof of Theorem 12.

Note that Proposition 14 already proves the same but uses polynomially many queries
to the set in \( \text{NP}(A) \) deciding inequivalence. We next extend the above idea to prove that,
for \( A \in \text{Full-P}/\log \), any set in \( \text{NP}(A) \) can be decided in polynomial time with queries to \( A \)
directly, in the presence of a set in (un-relativized) \( \text{NP} \).

**Theorem 16** Kolmogorov-easy circuit expressions for every set \( A \in \text{Full-P}/\log \) can be
learned in polynomial time with membership queries (to \( A \)) and an \( \text{NP} \) oracle.

**Proof:** The statement is to be interpreted as indicated earlier in this section: the learner’s
time bound depends on the actual constant factor of the advice length. We have a single
learning algorithm provided that this constant is given as input to it.

First we show the fact that \( \text{NP}(A) \subseteq \text{P}(A \oplus \text{SAT}) \). That is, \( \text{Full-P}/\log \) is in the first level
of the extended low hierarchy. There are alternative proofs of this fact, e.g. via instance
complexity (combine [4] and [5]).

Let \( A \) be a set in \( \text{Full-P}/\log \). That means
\[
\forall n \exists w_n \left( |w_n| \leq c \log n \right) \forall x \left( |x| \leq n \right) (x \in A \iff \langle x, w_n \rangle \in B)
\]
where \( B \in \text{P} \) and \( c \) is a constant. Let \( M \) be the machine that decides the set \( B \). Define the
set \( C \) as follows:
\[
C = \{ \langle x, v, w, 0^k \rangle \mid \exists z \mid x \leq k \text{ and } x \subseteq z \text{ and } M(\langle z, v \rangle) \neq M(\langle z, w \rangle) \}
\]
It is clear that \( C \) is in the class \( \text{NP} \). Suppose that \( L \) is an arbitrary set in \( \text{NP}(A) \); let us see
that \( L \in \text{P}(A \oplus \text{SAT}) \) using as oracle in \( \text{NP} \) the set \( C \). The idea is to look for some strings
that distinguish between advice words, with the help of oracle \( C \). When these strings are
known it is easy to discard the incorrect advice words, with queries oracle \( A \). Once a correct
advice word has been found, the decision on the input depends on the answer given by \( M \),
on the input itself, together with that correct advice word. As the number of all possible
advice words is not very large, this process can be done in polynomial time. The following
algorithm shows this fact:

```
input x;
take every possible advice word w of length bounded by \( c \log |x| \);
{there are a polynomial number in |x| of words w’s}
for all pairs of advice words \( \langle v, w \rangle \) do
    p := \lambda;
    if \( M(\langle p, v \rangle) \neq M(\langle p, w \rangle) \)
        then finished := true;
    else finished := false;
end if;
while not finished do
```

11
if \( \langle p0, v, w, 0^{|l|} \rangle \in C \)
then \( p := p0; \)
else if \( \langle p1, v, w, 0^{|l|} \rangle \in C \)
then \( p := p1; \)
else finished := true;
end if;
end if;
end while;

keep the word \( p \) associated to \( \langle v, w \rangle \) when this word exists;
end for;

for all pairs \( \langle v, w \rangle \) that become distinguished by their associated \( p \) do
discard \( v \) or \( w \) according to the answer of oracle \( A \) on query \( p \);
end for;

\{ Now any of the remaining advice words can be used to decide \( |x| \) \}
take any not discarded advice word \( w \);
accept \( x \) if and only if \( M(\langle x, w \rangle) \) accepts;

An alternative implementation would be to keep a “winning” advice word stored, while cycling over all of them. For each newcomer, the NP oracle plus possibly a membership query allows one to either check that they are equivalent, or find which one is wrong. In either case one is discarded.

Now a learner works as the above algorithm: given two potential advice words \( v, w \), tries to find a word on which they give different answers; this is where the NP oracle is used. Membership queries about the distinguishing words identify a single good advice. Then, applying the same construction as in Theorem 12, the Kolmogorov-easy circuit expression is produced.

It is worthwhile noting for later use that this algorithm can make the membership queries in a nonadaptive way, after all the queries to NP have given distinguishing strings for all pairs of non equivalent advice words. \( \square \)

The previous result has been expressed in terms of the concepts \((A)\) that the Kolmogorov-easy circuit expressions are representing. We can reformulate the above theorem focusing more on the learnability of the representation class than on the class of concepts Full-P/log.

**Corollary 17** If \( P = NP \) then KCE can be learned in polynomial time with membership queries.

Here, as before, we are slightly abusing terminology as described above in this section. The dependence of the learner on the constant comes now in two points: the length of advice words to cycle through and the precise polynomial time subroutine given by the hypothesis \( P = NP \) to solve what formerly were oracle queries to NP. This second dependence can be easily avoided, if necessary (or, to be more precise, reduced to the first one) by using a single fixed set \( B \) as polynomial time base machine for all sets \( A \). Any P-complete set will do, but the most natural one to use in our context would be, of course, \( CEVP \), as in Theorem 7.
It is natural to ask whether the implication in this corollary can be reversed, assuming unrelativized learnability and trying to obtain P=NP as a consequence. A corollary of our following result is that such a proof should not relativize.

Indeed, our next result reduces the strength of the complexity-theoretic hypothesis needed to obtain learnability, weakening the presence of an NP oracle to the so-called hypothesis $Q$ (See section 2 for the definition of $Q$).

**Theorem 18** KCE can be learned in polynomial time with membership queries if and only if $Q$ holds.

**Proof:** It is shown in Theorem 16 how to learn such expressions provided that the algorithm has access to a function satisfying the following specification: given a length $k$ in unary, and two advice words $u$ and $v$ of length $c \log k$ for use with a polynomial time machine $M$, output, if it exists, a word $z$ of length at most $k$ for which $M(\langle z, v \rangle) \neq M(\langle z, w \rangle)$. This is done through a prefix search with the help of the NP set $C$, but could be done otherwise; then membership queries discard all the wrong advice words and a circuit expression can be obtained from any of the good ones. Equivalently, one can see that argument as follows: given a length $k$ in unary, and Kolmogorov seeds of length $c \log k$ for two circuit expressions, output, if it exists, a word $z$ of length at most $k$ that is accepted by exactly one of the two expressions.

Now we prove that $Q$ implies that such a function can be computed in polynomial time. (The degree of the polynomial depends on the constant factor for the logarithmic bound on the complexity of the circuit expression, again.) First consider the following small change in the specification: $k$ is to be given in binary. Then it is obvious how to program a linear exponential time nondeterministic machine to find the required word, by guessing and checking it. From hypothesis $Q$, a function satisfying the new specification can be computed in deterministic linear exponential time. A preprocessor that compresses the unary encoding of $k$ into binary and then runs this exponential time (on the logarithmically shorter) input obtains, in polynomial time, the function needed in the algorithm of Theorem 16.

To prove the converse, we show that learnability of Kolmogorov-easy circuit expressions entails $Q$. Consider any nondeterministic machine $M$ working in linear exponential time on some input $x$. Applying the standard construction ([13], [20]) that proves the completeness of SAT for NP to $M$ and $x$ yields an exponentially long formula $\phi$ such that, first, it is satisfiable if and only if $M(x)$ accepts, and second, an accepting computation can be obtained with very small overhead from any satisfying assignment for $\phi$. We describe how to find one deterministically in exponential time: note, beforehand, that the formula is completely described by the code of the machine and the input, which is logarithmically small, and therefore is itself a Kolmogorov-easy circuit expression.

So simply run the learner on $0^{2n}$. It will work for polynomial time on that length, which is thus linear exponential time on the length of the input to the original machine, and will make membership queries, which in our case is tantamount to evaluating the formula on some assignments: it takes again linear exponential time on the input length.

Consider for a moment what would happen if the formula is unsatisfiable. In this case the behavior of the learner is completely determined, since all the membership queries will

---

[^2]: As reprinted in [27].
be answered with NO. Thus it must output some unsatisfiable formula. Indeed, it must come up with a formula equivalent to the learned one, since there are Kolmogorov-easy formulas of the same length that are unsatisfiable.

Hence the learner, trying to learn a satisfiable formula, must find out in polynomial time (and a fortiori polynomially many queries) on the length of the formula, that the formula it is trying to learning is not unsatisfiable. Thus it must notice the difference somehow with unsatisfiable formulas, and this can be done only through a positively answered membership query. This means that, if the formula is satisfiable, then one of the membership queries that we find will be necessarily answered yes: the simulation of the learner stops there since the objective, a satisfying assignment for the formula, has been found.

The statement can be refined by stratifying $Q$ into infinitely many hypothesis depending on the constant factors involved and analyzing their individual relationships with the learnability of $\text{KCE}_c$ for fixed $c$ (which is where the word “learnability” can be properly used). We do not pursue this study here.

This theorem, in a sense, weakens the oracle needed to run a polynomial time learning algorithm, as indicated in the previous theorem. Our characterization gives us the means of exploring to what extent it can be still weakened: to learn without auxiliary oracle establishes $Q$.

What about the following intermediate possibility: the oracle is still there but the access to it is restricted? It is easy to see that the use of the NP-oracle in the algorithm of Theorem 16 is adaptive, since it is used to compute a function with the above indicated specification. We have already mentioned that one can learn with non-adaptive use of the membership queries to the target concept. Can we learn under non-adaptive access to the NP oracle? This is to be understood as that all the queries to the NP oracle are listed down before making them; adaptive or nonadaptive membership queries to the target concept can be made both before and after the NP queries.

As we saw above, polynomial time learnability establishes $Q$. This intermediate, non-adaptive case however turns out to establish that $Q$ is equivalent to the equality $E=\text{NE}$.

**Theorem 19** Assume that a polynomial time learning algorithm exists for Kolmogorov-easy circuit expressions, with non-adaptive access to an oracle $A$ in $\text{NP}$. Then $E=\text{NE}$ implies $Q$ (i.e. $E=\text{NE}$ if and only if $Q$).

**Proof:** Consider the learner that is assumed to exist, and assume it first makes a few sequential membership queries to the target (phase 1), then prepares the list of parallel queries to NP, makes them, and then finally makes more sequential membership queries (phase 2) and stops (so there is nonadaptiveness in the NP queries only). Call the “risky path” the one on which all membership queries of phase 1 are answered NO.

The set $A'$ of all pairs $(0^n, x)$ such that $x \in A$ and $x$ is (non-adaptively) queried to its NP oracle somewhere on the risky path of the learner on input $0^n$ forms a set in NP. Moreover, all such pairs have logarithmic Kolmogorov complexity, because $n$ and the index of $x$ in the list of queries to the NP oracle suffice to describe them. (Non-adaptiveness is crucial here, as is the restriction to the risky path.) By standard complexity-theoretic arguments, $A'$ can be encoded into a tally set $T$ in NP, which is in $P$ under the hypothesis that $E=\text{NE}$.
Now run the learner on a specific CNF K-easy formula as in the proof of Theorem 18. If a solution is found in phase 1, then stop; else, we are on the risky path and can use the set $T$ to go on and complete the construction of a satisfying assignment, if one exists. Therefore $Q$ holds.

Note that all of the arguments used here do relativize, under the appropriate notion of relativized learning and Kolmogorov-easy circuit expressions. (Everything, including even the expressions, must be relativized, i.e. they are allowed to contain oracle gates). We get the following:

**Corollary 20** Under the appropriate relativization, Kolmogorov-easy circuit expressions can be learned in polynomial time with adaptive access to an NP oracle, but cannot be learned with non-adaptive access to the oracle (and in particular under no access to the oracle).

**Proof:** Impagliazzo and Tardos [17] have shown that there is a relativization such that E=NE but $Q$ fails. Under such a relativization, by Theorem 19, non-adaptive access to NP does not suffice to learn Kolmogorov-easy circuit expressions.

## 5 Easy circuit expressions from membership queries

The purpose of this section is to precisely characterize those concepts for which circuit expressions can be found in polynomial time using only membership queries, unconditionally and without any additional NP oracle. Note that to construct, in polynomial time, Kolmogorov-easy circuit expressions is equivalent to finding their logarithmically long seeds, since these can be found in polynomial time by exhaustive search.

**Theorem 21** The following facts are equivalent:

1. $A \in E(\text{Tally2})$.

2. $A$ is decided by a family of Kolmogorov-easy circuit expressions whose descriptions can be obtained in polynomial time using queries to $A$.

**Proof:** First we prove $(1) \Rightarrow (2)$.

Assume $A$ is a set in $E(\text{Tally2})$. That is, there exists a tally2 set $T$ fulfilling $A \in P(T)$ and $T \in P(A)$. Let $n_i$ and $n_k$ be the polynomials that bound the running time of the machines that query oracle $T$ and oracle $A$ respectively. $T \leq^{n_i} n_k$ suffices to decide which words of length $n$ are in $A$. Since $T$ is a tally2 set, all its words up to length $n_i$ have the form $0^{2^i}$ with $i \leq j \log n$. Therefore, $w_n = w_n^{1}w_n^{2}...w_n^{\log n}$, such that $w_n = 1$ if and only if $0^{2^i} \in T$, is an advice word for length $n$. Since $T \in P(A)$, there exists a polynomial-time algorithm that with input $n$ in unary, constructs the advice word $w_n$ querying $A$. Once $w_n$ is known, the algorithm is able to produce a circuit expression for $A \leq^{n}$ using the mechanism of Theorem 12.

Second, we show $(2) \Rightarrow (1)$. Let $A$ be a set recognized by a family of Kolmogorov-easy circuit expressions $\{E_n | n \in \mathbb{N}\}$. Assume that the logarithmically long seeds $w_n$ of the
expressions $E_n$ can be obtained in polynomial time by querying $A$. As we did in Theorem 12, we can keep only some selected $w_n$’s in order to encode them into a tally2 set.

We define the tally2 set $T$ containing only the information for seeds of length a double power of 2, and prove that $A \in E(T)$. These seeds are those corresponding to lengths $2^m$, $2^i \ldots 2^{i_m}$.

The length of each seed $w_{2^m}$ is exactly $2^m c$ (otherwise, append a suffix $10^*$). So the tally2 set is the following:

$$T = \{0^p(x_{i \leq m-1}2^{c}) | 1 \leq p \leq 2^m c \text{ and the } p\text{-th bit of } w_{2^m} \text{ is } 1\}$$

It is easy to see that $A \in P(T)$. On input $x$, find an integer $m$ such that $2^{m-1} < |x| \leq 2^m$.

Next for each value of $p$ from 1 to $2^m c$, ask whether $0^p(x_{i \leq m-1}2^{c}) \in T$ and, in this way, obtain all the bits of the seed $w_{2^m}$, which can now be used to decide whether $x \in A$ in polynomial time. Thus, $A \in P(T)$.

Conversely, $T \in P(A)$. To know whether $0^p \in T$, look for $m$ such that $c(2^m - 1) < n \leq c(2^{m+1} - 1)$. Thus, $2^m = 2c(2^{m-1} + p)$ with $1 \leq p \leq c2^m$. After that, construct $w_{2^m}$ querying oracle $A$ in order to check whether the $p$-th bit is 1. Hence $T \in P(A)$ and $A \in E(Talley2)$.

A similar result was already proven in [8], although it was presented in terms of logarithmic advice words. Actually both formulations turn out to be the same because, in this context, the descriptions of Kolmogorov-easy circuit expressions play the same role as the logarithmic advice words.

From Theorem 18 we obtain an improvement on a previous result from [8], where it was shown that $P = NP$ implies $E(Talley2) = P(Talley2)$.

**Corollary 22** $E(Talley2) = P(Talley2)$ if and only if $Q$ holds.

**Proof:** $(\Leftarrow)$ We only need to show that $A \in P(Talley2)$ implies $A \in E(Talley2)$. Assume $A \in P(Talley2)$. By Theorem 11 and Theorem 12 $A$ is decided by a family of Kolmogorov-easy circuit expressions. Now assume that $Q$ holds. By Theorem 18, KCE is learned in polynomial time with membership queries. Inspection of the proof reveals that the algorithm is able to produce the description instead of the Kolmogorov-easy circuit expression itself. Therefore for each $c$, the descriptions for Kolmogorov-easy circuit expressions in KCE, are found in polynomial time with membership queries. By Theorem 21, this implies that $A \in E(Talley2)$.

$(\Rightarrow)$ By Theorem 21 $E(Talley2) = P(Talley2)$ implies that KCE is learnable in polynomial time with membership queries, and thus by Theorem 18 $Q$ holds.

---

## 6 Circuit expressions with polylogarithmic seeds

This section studies how we can relax the condition of Kolmogorov-easy circuit expressions on which all the constructions are based in previous sections. Unfortunately, we cannot extend all the results to bounds larger than logarithmic. But, for certain reasonable conditions on the bounds, some parts carry through; and, concentrating on polylogarithmic bounds, there are alternative characterizations that extend Theorem 21.
The fact that it is possible to find a family of advice words for each set in Full-P/log with the property that all of them are prefixes of a single infinite word was shown in [8] (announced previously in [9]). The same property holds in the case of advice words of larger size than logarithmic, and it will be the main tool to extend Theorem 21 to the polylog case. First, we define formally a extended version of the class Full-P/log.

**Definition 23** Given a particular family of functions \( \mathcal{F} \)

1. A set \( A \) is in Full-P/\( \mathcal{F} \) if there exists a function \( f \in \mathcal{F} \) such that:

\[
\forall n \exists w_n (|w_n| \leq f(n)) \forall x (|x| \leq n) x \in A \iff \langle x, w_n \rangle \in B
\]

where \( B \in \text{P} \).

2. A set \( A \) is in Pref-P/\( \mathcal{F} \), if \( A \) is in Full-P/\( \mathcal{F} \) via an infinite sequence of advice words \( w_n \) having the additional property that for all \( n \leq m \), \( w_n \) is a prefix of \( w_m \).

Under some conditions, both Full-P/\( \mathcal{F} \) and Pref-P/\( \mathcal{F} \) denote the same nonuniform class.

**Theorem 24** If \( f(n) \) is monotone and can be calculated in polynomial time, then Full-P/\( \text{O}(f(n)) = \text{Pref-P/O}(f(n)) \).

**Proof:** The nontrivial inclusion is from left to right. Let \( A \in \text{Full-P/O}(f(n)) \), so that there exists a Turing machine \( M \) that decides \( A \) in polynomial time, with the help of advice words \( \{w_n\}_{n \in \mathbb{N}} \). There exists a function \( h(n) \in \text{O}(f(n)) \) such that each \( w_n \) has the length bounded by \( h(n) \). Without loss of generality we assume that \( h \) is monotone too. We have to construct an infinite sequence, in such a way that the prefixes of length \( \text{O}(f(n)) \) can be used as well as advice words for \( A \). Similarly to the proof of Theorem 21 we construct an infinite sequence by concatenating some selected \( w_n \)'s.

Since \( h \) is monotone, it is possible to find for each \( n \) a number \( j_n \) such that \( 2^{j_n - 1} \leq h(n) \leq 2^{j_n} - 1 \); moreover, given \( n \), the search of \( j_n \) is done in polynomial time. Remark that eventually many \( n \)'s are associated to the same \( j_n \). Let \( p(j) \) be the following function:

\[
p(j) = \max\{n \mid h(n) \leq 2^j - 1\}
\]

When this maximum does not exist, necessarily all values for \( n \) but finitely many of them have the same image by \( h \). In this case choose one of these possible values for \( p(j) \). Construct an infinite sequence \( \beta \) including all \( \{w_{p(j)}\}_{j \in \mathbb{N}} \). In order to get a good manipulation of the sequence, we construct \( \beta \) step by step, appending a power of two many bits each time. Namely \( \beta = t_0 t_1 t_2 \ldots t_n \ldots \) where, for each \( i \), \( t_i = w_{p(j)}10^{2^i - |w_{p(j)}| - 1} \). Therefore

\[
\sum_{i=0}^{j} |t_i| = \sum_{i=0}^{j} 2^i = 2^{j+1} - 1
\]

The minimum prefix of \( \beta \) containing the information of the advice \( w_{p(j)} \) is exactly \( \beta_1 2^{j+1} - 1 \). Then \( A \) is in Pref-P/\( \text{O}(f(n)) \) because the following algorithm decides whether a word \( x \) is in \( A \).
input $x$;

$n := |x|$;

Find $j$ such that $2^{j-1} \leq h(n) \leq 2^j - 1$;

From $\beta_{1,2^{j+1}-1}$ obtain $\beta_{2,2^{j+1}-1}$;

Discard the $10^*\text{ tail}$, obtaining advice word $w$;

Simulate $M((x, w))$ and accept if and only if $M$ accepts.

Exactly $2^{j+1} - 1$ bits from $\beta$ are used to decide whether $x$ belongs to $A$. From the fact that $2^{j-1} \leq h(n)$, we obtain that $j \leq \log(h(n)) + 1$, so $2^{j+1} - 1$ is bounded by $4h(n)$. Therefore, for each $n$ it suffices to know the first $4h(n)$ bits of $\beta$ in order to decide whether the strings of length $n$ belongs to $A$, that means, $A \in \text{Pref-P/O}(f(n))$. \qed

We will focus next on the nonuniform classes defined from polylog advice words, that is, $\text{Full-P/O}((\log n)^i)$ with $i$ a constant. The next question we address is how to extend the characterization of the sets with Kolmogorov-easy circuit expressions learnable only from membership queries, to not-so-easy (e.g. polylog-easy) circuit expressions. Observe that nowhere in the class $E(\text{Tally2})$ there appears explicitly a logarithm that could be changed into a polylog function. Actually, as it turns out, there is a nice reason for this.

Indeed, as we shall see shortly, there is a marked difference between the logarithmic and the polylogarithmic case. Both allow for a characterization in terms of polynomial-time Turing degrees of tally sets. In the case discussed in the previous section, the tally sets used were defined by a “qualitative” condition that the words were only of certain fixed lengths, namely powers of 2. When we move to polylog functions, the characterization no longer has this flavor, but it instead becomes purely “quantitative”: it corresponds to tally degrees of polylog density. Moreover, the exponent of the density tightly corresponds to the exponent of the complexity of the circuit expressions, modulo a log factor; and this log factor precludes the use of this characterization for the logarithmic case. So the theorem of the previous section cannot be obtained as a particularization of the present one. (We do not rule out the possibility that a more general fact can be shown from which both could be consequences; we are not aware of such a fact.)

Let us introduce some notation to identify the class of tally sets of density bounded by a function of the class $\mathcal{F}$. We denote it as $\mathcal{F}$-$\text{Tally}$.

**Definition 25** Let $f$ be a nondecreasing function, $f : \mathbb{N} \rightarrow \mathbb{N}$. A set $S \subseteq \Sigma^*$ is $f(n)$-dense or has $f(n)$ density if $|S^{\leq n}| < f(n)$ for all $n$.

If $F$ is a class of functions, we say $S$ is $F$-dense if $S$ is $f$-dense for some $f \in F$.

$\mathcal{F}$-$\text{Tally}$ is the class of all $F$-dense tally sets.

Polylogarithmic nonuniformity is related to polylogarithmic-dense tally sets used as oracles, as we show in the following theorem.

**Theorem 26** For all $i \geq 1$, the following holds:

$\text{Full-P/O}((\log n)^{i+1}) = \text{P(O((\log n)^i)}$-$\text{Tally})$. 

18
Proof: We indicate first why \( \text{P}(O((\log n)^i)\text{-Tally}) \subset \text{Full-P}/O((\log n)^{i+1}) \). Let \( A \) be a set in \( \text{P}(O((\log n)^i)\text{-Tally}) \) via a tally set \( L \). In polynomial time, at most a polynomially long, say \( n^c \), initial part of \( L \) is accessible, and we need to know the \( O((\log n^c)^i) \) strings in \( L \) up to that length. Each string can be identified by its length with at most \( c \log n \) bits. If we want to see whether an input \( x \) belongs to the set \( A \), then we use the information about \( L^{\leq n^c} \) encoded into \( c \log |x| \cdot d((\log |x|)^i) \leq c((\log |x|)^{i+1} \text{ bits, for appropriate constants } c, d, e. \) These bits constitute our advice word.

Conversely, we will show that \( \text{Full-P}/O((\log n)^{i+1}) \subset \text{P}(O((\log n)^i)\text{-Tally}) \) using the fact that \( \text{Full-P}/O((\log n)^{i+1}) \subset \text{Pref-P}/O((\log n)^{i+1}) \). Consider a set \( A \in \text{Full-P}/O((\log n)^{i+1}) \) that has a sequence of advice words \( \{w_n\}_{n \in \mathbb{N}} \). Each advice word has length \( d((\log n)^{i+1}) \) (for some constant \( d \)). By Theorem 24 there exists a single infinite sequence \( \beta \) whose prefixes contain enough information about \( \{w_n\}_{n \in \mathbb{N}} \). Namely, each prefix of \( \beta \) that has length \( 4d((\log n)^{i+1}) \) suffices to produce a string that can be used as advice word of \( A \) for length \( n \).

The proof is divided into two parts:

1. We encode \( \beta \) into a tally set of the indicated density in the following way: for all but finitely many \( n \)'s, the information \( \beta_{1:4dn^{i+1}} \) is kept into the tally set up to the word \( 0^{2^i} \).

2. We show that \( A \) is Turing reducible to this tally set.

The encoding scheme of part 1 is as follows. We define a tally set \( L \) of \( k((\log n)^i \text{ density, where } k \text{ is a constant. In the interval between } 0^{2^i} \text{ and } 0^{2^{i+1}}, \text{ we will add at most } k(n+1)^i - kn^i \text{ new words belonging to } L \). That means we will add at most \( O(n^{i-1}) \) new elements, because:

\[
k(n+1)^i - kn^i = k\left(\binom{i}{0}n^i + \binom{i}{1}n^{i-1} + \ldots + \binom{i}{i}n^{i-1}\right) - kn^i \leq rn^{i-1}
\]

for some constant \( r > 1 \). We have the freedom to choose these strings among the \( 2^n \) words in the interval. To this end we divide the \( 2^n \) words into \( n^{i-1} \) parts. Each one has the following number of words:

\[
\mathcal{N} = \left\lfloor \frac{2^n}{rn^{i-1}} \right\rfloor
\]

Now, we can interpret each part as a digit in base \( B = \mathcal{N} + 1 \), depending on where the only word of that part is, and encode a number with \( rn^{i-1} \) such digits. With this strategy, the number of bits we can encode is \( rn^{i-1} \) times the length of the binary representation of the digits in base \( B \):

\[
\prod_{i=1}^{n} \log \mathcal{N} = rn^{i-1} \cdot \log(\frac{2^n}{rn^{i-1}})
\]

Denote this number by \( J \). Our goal is to store the information \( \beta_{1:4dn^{i+1}} \) up to the word \( 0^{2^n} \) of the tally for almost every \( n \). The next calculations show that there is enough room to encode \( \beta \) in this way in the tally \( L \). From the length \( 2^n \) to \( 2^{n+1} \), the number of bits \( \beta \) has had to grow is:

\[
4d(n+1)^{i+1} - 4dn^{i+1} = 4d\left(\binom{i+1}{0}n^{i+1} + \ldots + \binom{i+1}{i+1}\right) - 4dn^{i+1} \leq cn^i
\]
For $c$ another constant. It is easy to see that the number $cn^i$ is smaller than $J$ when the constant $r$ (in $J$) is conveniently chosen (Note that $r$ depends on $k$ and we can choose $k$ and hence $r$ as we wish).

$$J = rn^{i-1} * \log\left(\frac{2^n}{rn^{i-1}}\right) = rn^{i-1} * (n - \log r - (i - 1) \log n)$$

Setting $\log r$ to $c'$:

$$J = rn^i - c'rn^{i-1} - (i - 1)rn^{i-1} \log n$$

Choosing $r$ as $2c$:

$$cn^i \leq J \iff cn^i \leq 2cn^i - c'2cn^{i-1} - (i - 1)2cn^{i-1} \log n$$

Rearranging terms we get:

$$c'2cn^{i-1} + (i - 1)2cn^{i-1} \log n \leq cn^i$$

which is true for all but finite many $n$'s.

Regarding part 2, we have to show that $A \in \text{P}(L)$. The following algorithm suffices to decide whether a word $x$ belongs to $A$ querying oracle $L$.

**input** $x$;

$n := |x|$

Querying oracle $L$ up to $2^n$ recover $\beta_{1:4d(\log n)^i+1}$;

Look for the maximum value $m$ such that $d(\log n)^i+1 \leq 2^m$;

From $\beta_{1:2m+1:-1}$ obtain $\beta_{2m:-2m+1:-1}$;

Discard the $10^*$ tail, obtaining advice word $w$;

Decide $x$ with the help of the advice word $w$.

The lengths of the queries are smaller than the size of the input, and only a linear amount of queries is made to oracle $L$. The process of writing down the string $\beta_{1:4d(\log n)^i+1}$ from the information $\chi^{L\leq n}$ can be done in polynomial time. The next steps in the algorithm are similar to those given in the process of recovering the advice for $n$ from the infinite sequence $\beta$ (Theorem 24).

As in the case of Full-P/log, it is possible to characterize the classes with polylogarithmic advice words in terms of circuit expressions:

**Theorem 27** For all $i \geq 2$, the following classes coincide:

1. Full-P/O((log $n$)$^i$).

2. The class of sets $A$ that have circuit expressions $\{E_n\}_{n \in \mathbb{N}}$ such that $E_n \in K[O((\log n)^i), \text{poly}]$ for all $n$.

**Proof:** A set $A$ is in Full-P/O((log $n$)$^i$) if

$$\forall n \exists w_n \ (|w_n| \leq c(\log n)^i) \ \forall x \ (|x| \leq n) \ (x \in A \iff \langle x, w_n \rangle \in B)$$
where $B \in P$ and $c$ is a constant.

The proof of $(1) \Rightarrow (2)$ is again the construction of the circuit expression from a particular length $n$, the advice word for $n$, and the program to generate the standard boolean circuit associated to that length (see Theorem 12).

In order to prove $(2) \Rightarrow (1)$, the seeds of polylogarithmic size are used as advice words. As in the logarithmic case, not all of them are selected, as a result of the problem that the advice words (in this case the seeds) can be used by very small lengths. Due to this fact, in the process of selecting them, we use again the same technique as in Theorems 12, 21, and 24.

Suppose the seeds are $w_1, w_2, \ldots, w_n, \ldots$ and that each one has length $c(\log n)^i$ for some constant $c$. Keep only the seeds $\{w_{p(j)}\}_{j \in \mathbb{N}}$, where $p(j)$ is the function defined as before:

$$p(j) = \max\{n \mid c(\log n)^i \leq 2^j - 1\}$$

Actually, we assume that every $w_{p(j)}$ is exactly of size $2^j$, since we can append $10^{2^j - b_{p(j)} - 1}$ to the true seed $s_{p(j)}$.

The advice word for each length $m$ is constructed looking for the number $j$ such that:

$$2^{j-1} \leq c(\log m)^i \leq 2^j - 1$$

Once found the number $j$, the advice word for length $m$ is

$$w'_m = w_{p(1)}w_{p(2)} \cdots w_{p(j)}$$

Following the same steps as in Theorems 12, 21, 24, it is easy to see that every $w'_m$ is a correct advice word and has length bounded by $O((\log m)^i)$. □

Combining these results, we can identify which languages have polylogarithmic circuit expressions that can be learned in polynomial time using only membership queries.

**Theorem 28** For all $i \geq 1$, the following facts are equivalent:

1. $A \in E(O((\log n)^i)\text{-Tally})$.

2. $A$ is decided by a family of polynomial-size circuit expressions $E_n$ in $K[O((\log n)^{i+1}), \text{poly}]$, whose descriptions are learnable in polynomial time from membership queries.

**Proof:** First let us see that $(1) \Rightarrow (2)$. Assume $A$ is a set in $E(O((\log n)^i)\text{-Tally})$. The advice words for each length can be constructed in polynomial time querying $A$. In order to do this, use the fact that for some tally set $T$ of density $O((\log n)^i)$, $A \in P(T)$ and $T \in P(A)$. $T$ up to a length polynomial in $n$ suffices to decide the words of $A$ that have length $n$. Since the number of strings in $T$ is bounded by $O((\log n)^i)$, this encoding can be done in $O((\log n)^{i+1})$ bits. Actually, these encodings are the advice words and they are produced in polynomial time with membership queries.

Regarding the converse, we sketch only the main ideas to see that $(2) \Rightarrow (1)$. The proof follows three steps.
1. Construction of an infinite advice sequence $w$ from the seeds $w_n$ of the circuit expressions for $A$. This process is exactly the mechanism explained in Theorem 24.

2. Codification of $w$ into a tally set $T$ of density $O((\log n)^i)$, as was done in Theorem 26. Therefore $A \in \text{P}(T)$.

3. Justification that $T \in \text{P}(A)$. This is easy to see from the construction in previous steps, and the hypothesis that there exists an algorithm that learns all $w_n$’s in polynomial time with membership queries.

\[\square\]

**Acknowledgments**

We would like to thank David Guizarro and especially Ricard Gavaldà for many technical discussions in which they suggested interesting ideas, some of which appear in this paper.

**References**


