Proceedings of Machine Learning Research vol 313:1–19, 2026 37th International Conference on Algorithmic Learning Theory

# On the Hardness of Learning Regular Expressions

**Idan Attias** IDANATTIAS88@GMAIL.COM
*University of Illinois at Chicago and Toyota Technological Institute at Chicago*

**Lev Reyzin** LREYZIN@UIC.EDU
*University of Illinois at Chicago*

**Nathan Srebro** NATI@TTIC.EDU
*Toyota Technological Institute at Chicago*

**Gal Vardi** GAL.VARDI@WEIZMANN.AC.IL
*Weizmann Institute of Science*

**Editors:** Matus Telgarsky and Jonathan Ullman

## Abstract

Despite the theoretical significance and wide practical use of regular expressions, the computational complexity of learning them has been largely unexplored. We study the computational hardness of improperly learning regular expressions in the PAC model and with membership queries. We show that PAC learning is hard even under the uniform distribution on the hypercube, and also prove hardness of distribution-free learning with membership queries. Furthermore, if regular expressions are extended with complement or intersection, we establish hardness of learning with membership queries even under the uniform distribution. We emphasize that these results do not follow from existing hardness results for learning DFAs or NFAs, since the descriptive complexity of regular languages can differ exponentially between DFAs, NFAs, and regular expressions.

**Keywords:** Regular Expressions, PAC Learning, Membership Queries, Computational Hardness, Descriptional Complexity

## 1. Introduction

What is the computational complexity of learning a regular expression (RE) of length at most $s$ over $\{0,1\}^n$, or over some other alphabet $\Sigma^n$, as a function of $n, s$, and $|\Sigma|$? Can this be done in time poly$(n, s)$? Surprisingly, despite the importance of regular expressions, this question has been left open and not explicitly addressed in the literature. In this paper, we establish hardness of learning regular expressions in the Probably Approximately Correct (PAC) and Membership Queries (MQ) models, both under arbitrary input distributions and under the uniform distribution. Our results are summarized in Table 1.

Learning regular languages has been extensively explored in the literature. In particular, some of the earliest results, both about proper learning in an inductive setting (Gold, 1978; Pitt and Warmuth, 1993) and cryptographic hardness of improper PAC learning (Kearns and Valiant, 1994), established the hardness of learning Deterministic Finite Automata (DFA). DFAs and REs are indeed equivalent in that both exactly characterize regular languages (Kleene, 1956): any DFA has an equivalent RE, and every RE has an equivalent DFA. Why can't we then conclude that since DFAs are hard to learn, i.e., regular languages are hard to learn, then REs are hard to learn? The important point is that when we say DFAs or REs are easy or hard to learn, we mean that it is easy or hard to learn languages with *succinct* DFAs or REs. But even though every DFA has an equivalent RE and vice versa, the

© 2026 I. Attias, L. Reyzin, N. Srebro & G. Vardi.

conversion may require exponential (or even super-exponential) blowups. These gaps are discussed in Section 2.1 and summarized in Table 2. Therefore, even if we cannot learn in time polynomial in the size (or number of states) of a DFA, this does not contradict the possibility of learning in time polynomial in the length of an RE. Indeed, as we can see in Table 1, there are differences in the complexity of learning DFAs vs. REs. In particular, for MQ learning, it is possible to learn in time polynomial in the size of the DFA, but not polynomial in the length of the RE.

Table 1: Tractability of learning regular expressions and automata. Our new hardness results are in bold font.

| | **PAC** | | **PAC + MQ** | |
|---|---|---|---|---|
| **Representation complexity** | Dist.-free | Uniform dist. | Dist.-free | Uniform dist. |
| poly$(n)$ size DFA | Hard[A,B,C,D] | Hard[C] | Tractable | Tractable |
| poly$(n)$ size NFA | Hard[A,B,C,D] | Hard[C] | Hard[A,**B,C,D**] | ? |
| poly$(n)$ length RE (with only $\vee, \cdot, ^\star$) | **Hard**[A,B,C,D] | **Hard**[C] | **Hard**[B,C,D] | ? |
| poly$(n)$ length RE with intersection | **Hard**[A,B,C,D] | **Hard**[A,C] | **Hard**[A,B,C,D] | **Hard**[A] |
| poly$(n)$ length RE with complement | **Hard**[A,B,C,D] | **Hard**[A,C] | **Hard**[A,B,C,D] | **Hard**[A] |

[A] Under one of the assumptions: RSA, factoring Blum integers, or quadratic residues.

[B] Assumption: random $k$-SAT.

[C] Assumption: local-PRG.

[D] Assumption: sparse learning-parity-with-noise.

- DFA: Kearns and Valiant (1994) showed distribution-free PAC hardness under Assumption A; Daniely and Shalev-Shwartz (2016) established it under Assumption B; and Daniely and Vardi (2021) under Assumption C. Moreover, Daniely and Vardi (2021) proved PAC hardness on the uniform distribution (again under Assumption C). There is a PAC-preserving reduction from DNFs to DFAs (Pitt and Warmuth, 1990); therefore, the hardness of learning DNFs under Assumption D proved by Applebaum et al. (2025) immediately implies the hardness of learning DFAs. In contrast, DFAs are tractably learnable with membership queries via the $L^\star$ algorithm of Angluin (1987).
- NFA: PAC hardness follows directly from the hardness of learning DFAs. Moreover, for distribution-free learning with membership queries, Angluin and Kharitonov (1995) proved hardness under Assumption A, and since REs are expressible using NFAs with polynomial blowup, our results for REs imply hardness under Assumptions B,C,D.
- Plain RE: We establish distribution-free hardness in PAC under assumptions B,C,D (see Theorem 4.2). Angluin et al. (2013) proved hardness with a *non-binary* alphabet of a subclass of regular expressions, called shuffle ideals, under assumption A. We claim that their construction can be modified to work for plain REs and the binary alphabet, see discussion in Section 4.3.
  In Theorem 4.4 we prove PAC hardness under the uniform distribution, relying directly on Assumption C. In Theorem 5.1 we show distribution-free hardness of learning with queries under Assumptions B,C,D (and the existence of non-uniform one-way functions).
- RE with intersection or complement: First, all hardness results for plain REs also apply here. Second, in Theorem 5.3, we prove hardness with membership queries on the uniform distribution under Assumption A.
- Open Questions: As far as we are aware, it remains unknown whether there is a tractable algorithm for learning plain REs or NFAs with membership queries under the uniform distribution.

Learning regular languages, and REs in particular, is of practical as well as conceptual importance. Many rules in everyday data processing, manipulation, filtering, and analysis tasks are specified using regular expressions (e.g. validating numbers, extracting street names from addresses, or rewriting names as last–comma–first). REs are at the core of many languages and tools focused on data processing (e.g., grep, awk, and Perl), and are important elements of many others (e.g., Excel, Python, and almost all modern programming languages and shells). Thinking of machine learning as replacing expertly specified rules with learning from examples, being able to learn an RE from examples is thus a very natural task, and indeed a practical feature (e.g., given a column of addresses and a few examples of the house number, the machine should learn to extract the rest). Importantly, in all these languages, regular languages are specified by short REs, not DFAs. In addition to the classical REs as specified by Kleene, which support union, concatenation, and Kleene-star operations, several extensions to REs have also been suggested and are commonly supported by programming languages and libraries. We discuss these in Section 2.

Learning Regular Languages has also recently received renewed attention, with the study of what Regular Languages transformers can represent and learn (e.g., Strobl et al. (2024) and the references therein). Indeed, this has been the direct impetus for our study. Part of the goal of this paper is to emphasize the difference between different representations of Regular Languages, and in particular between DFAs and REs, to highlight the importance of studying learnability quantitatively as a function of a specific description language, and to encourage the study of learnability in terms of Regular Expression length.

## 2. REs and Regular Languages

We focus on the binary alphabet $\Sigma = \{0, 1\}$. $\Sigma^\star$ defines the set of all binary strings, where $\Sigma^\star = \cup_{n\in\mathbb{N}}\Sigma^n$, with $\mathbb{N}$ including zero.

**Regular Expressions (RE).** The set of regular expressions over an alphabet $\Sigma$, denoted $\mathsf{RE}(\Sigma)$, is the minimal set defined as follows: $\emptyset$, $\epsilon$, and every symbol $a \in \Sigma$ belong to $\mathsf{RE}(\Sigma)$. If $R_1, R_2 \in \mathsf{RE}(\Sigma)$, then the expressions $(R_1 \vee R_2)$ (union), $(R_1R_2)$ (concatenation, also denoted $(R_1 \cdot R_2)$), and $(R_1)^\star$ (Kleene star) are also in $\mathsf{RE}(\Sigma)$.

For $R \in \mathsf{RE}(\Sigma)$ the language it represents, written $L(R) \subseteq \Sigma^\star$, is defined inductively: $L(\emptyset) = \varnothing$, $L(\epsilon) = \{\epsilon\}$, $L(a) = \{a\}$ for $a \in \Sigma$. $L(R_1 \vee R_2) = L(R_1) \cup L(R_2)$, $L(R_1R_2) = \{xy : x \in L(R_1), y \in L(R_2)\}$, and $L(R^\star) = \bigcup_{k\in\mathbb{N}} L(R)^k$, where $L(R)^0 = \{\epsilon\}$ and $L(R)^{k+1} = L(R)^kL(R)$. A language $L \subseteq \Sigma^\star$ is called *regular* if $L = L(R)$ for some $R \in \mathsf{RE}(\Sigma)$. We use the shorthand $\mathsf{RE}$ for $\mathsf{RE}(\Sigma)$.

**Extended Regular Expressions.** Having fixed the standard syntax and semantics of regular expressions, we now extend the language by adding new standard operators.

*Intersection operator:* The set $\mathsf{RE}(\cap)$ is obtained by closing RE under intersection: if $R_1, R_2 \in$ RE$(\cap)$, then $(R_1 \wedge R_2) \in$ RE$(\cap)$. The semantics is defined by $L(R_1 \wedge R_2) = L(R_1) \cap L(R_2)$.

*Complement (negation) operator:* The set RE$(\neg)$ is obtained by closing RE under complement: if $R \in$ RE$(\neg)$, then $(\neg R) \in$ RE$(\neg)$. The semantics is defined by $L(\neg R) = \Sigma^\star \setminus L(R)$.

*Counting operator:* We also extend the language with a bounded-iteration (counting) operator, an operator supported in practice by tools such as egrep (Hume, 1988), Perl regular-expression syntax (Wall et al., 1999), and the XML Schema specification (Sperberg-McQueen and Thompson,

2005). For a regular expression $R$ and an integer $k \in \mathbb{N}$, we write the counting expression as $R^k$. Its semantics is given by $L(R^k) = L(R)^k$, that is, $R^k$ matches any word that can be decomposed into $k$ contiguous factors, each belonging to $L(R)$. We denote the resulting class of *counting regular expressions* by $\mathsf{RE}(\#)$. For related variants of the definition, succinctness results, and associated computational problems, see Meyer and Stockmeyer (1972); Kilpeläinen and Tuhkanen (2003); Gelade et al. (2012); Gelade (2010). We use the notation $R^k$ as shorthand even when explicit counting is not permitted. In such cases, it should be understood as repeated concatenation. When counting is allowed, the encoding simply becomes more succinct.

**Size of regular expressions.** We define the size of a (possibly extended) regular expression $R$, denoted by $|R|$, as the total number of atomic and operators that appear in the concrete syntax of $R$: $|\varnothing| = 1$, $|\varepsilon| = 1$, and $|a| = 1$ for $a \in \Sigma$. $|R_1 R_2| = |R_1| + |R_2|$, $|R_1 \vee R_2| = |R_1| + |R_2| + 1$, $|r^\star| = |r| + 1$, $|R_1 \wedge R_2| = |R_1| + |R_2| + 1$, $|\neg R| = |R| + 1$, and $|r^k| = |r| + \lceil \log(k) \rceil$, where counting is allowed and $k$ is encoded in binary, and otherwise $|r^k| = k|r|$. For alternative size measures (all equivalent up to a polynomial factor) and detailed succinctness results, see Ellul et al. (2005); Gelade (2010); Gruber and Holzer (2015).

**DFAs and NFAs.** A Nondeterministic Finite Automaton (NFA) is a tuple $A = (\Sigma, Q, Q_0, \delta, F)$, where $\Sigma$ is a finite alphabet, $Q$ is a finite set of states, $Q_0 \subseteq Q$ is the set of initial states, $\delta : Q \times \Sigma \to 2^Q$ is the transition function, and $F \subseteq Q$ is the set of final states. Given a word $w = \sigma_1 \cdots \sigma_\ell \in \Sigma^*$, a run of $A$ on $w$ is a sequence of states $r = q_0, \ldots, q_\ell$ such that $q_0 \in Q_0$ and $q_{i+1} \in \delta(q_i, \sigma_{i+1})$ for all $i \geq 0$. The run $r$ is accepting if $q_\ell \in F$. We say that $A$ accepts $w$ if it has an accepting run on $w$, and we denote by $L(A)$ the language of $A$, namely the set of words it accepts. When $|Q_0| = 1$ and $|\delta(q, \sigma)| \leq 1$ for all $q \in Q$ and $\sigma \in \Sigma$, then $A$ is deterministic. In this case we say that $A$ is a Deterministic Finite Automaton (DFA).

### 2.1. Equivalence and gaps between REs and automata

In this section, we review the vast literature on the succinctness of representation among different variants of REs, DFAs, and NFAs, see Table 2. A central claim of this paper is that the *description length* of objects representing regular languages plays a crucial role in determining their *tractable learnability*.

We compare these models by analyzing their description lengths, i.e., the length of a binary string encoding the object. For DFAs with $q$ states (over a fixed alphabet), the description length is upper bounded by $O(q \log q)$. For NFAs with $q$ states, a dense encoding of transition sets yields $O(q^2)$ bits. Although one may alternatively take the number of states as the size parameter (a common convention), this differs only by a polynomial factor. Importantly, the choice of size measure affects the analysis of conversions: for example, the transformation from a DFA to an NFA is linear in the number of states, whereas when measured in terms of description length the target can scale as $\Theta(q^2)$ rather than $\Theta(q \log q)$.

For regular expressions, the description length is defined as the size of the expression, namely the number of symbols and operators it contains. Parentheses can be disregarded, as they only affect the size by a constant multiplicative factor.[1] Thus, for a constant-size alphabet $\Sigma$ and a constant operator set $\Gamma$ (including $\varepsilon$ and $\emptyset$), an RE of length $\ell$ has description length $\ell \cdot \lceil \log\left(|\Sigma| + |\Gamma| + 2\right) \rceil =$

1. For a discussion of size definitions and their implications see Ellul et al. (2005); Gelade (2010); Gruber and Holzer (2015).

Table 2: Succinctness/translation blow-ups among DFA, NFA, RE, RE($\cap$), and RE($\neg$).

| **Source → Target** | DFA | NFA | RE | RE($\cap$)† | RE($\neg$)† |
|---|---|---|---|---|---|
| DFA (description length or #states) | – | poly($n$) | $2^{\Theta(n)}$ Ehrenfeucht and Zeiger (1976) Gruber and Holzer (2015) | $2^{O(n)}$ | $2^{O(n)}$ |
| NFA (description length or #states) | $2^{\Theta(n)}$ Rabin and Scott (1959) Moore (1971); Meyer and Fischer (1971) | – | $2^{\Theta(n)}$ Gruber and Holzer (2015) Ehrenfeucht and Zeiger (1976) | $2^{O(n)}$ | $2^{O(n)}$ |
| RE (length) | $2^{\Theta(n)}$ Gruber and Holzer (2015) Ehrenfeucht and Zeiger (1976) | poly($n$) Thompson (1968) Gruber and Holzer (2015) | – | poly($n$) | poly($n$) |
| RE($\cap$) (length) | $2^{2^{\Theta(n)}}$ Gelade (2010) | $2^{\Theta(n)}$ Gelade (2010) | $2^{2^{\Theta(n)}}$ Gelade and Neven (2012) | – | poly($n$) |
| RE($\neg$) (length)* | $\underbrace{2^{2^{\cdot^{\cdot^{2}}}}}_{\Theta(n)}$ Dang (1973) Stockmeyer and Meyer (1973) | $\underbrace{2^{2^{\cdot^{\cdot^{2}}}}}_{\Theta(n)}$ Dang (1973) Stockmeyer and Meyer (1973) | $\underbrace{2^{2^{\cdot^{\cdot^{2}}}}}_{\Theta(n)}$ Dang (1973) Stockmeyer and Meyer (1973) | $\underbrace{2^{2^{\cdot^{\cdot^{2}}}}}_{O(n)}$ | – |

* Depth-sensitive succinctness for conversions from RE($\neg$) to RE/DFA/NFA:
The complexity of the conversion grows with the complement-nesting depth of the regular expression. Thus, with unbounded depth the blow-up is non-elementary. For a single negation (depth 1), translating back to a plain RE already requires $2^{2^{\Theta(n)}}$ symbols (Gelade and Neven, 2012). See also Gelade (2010) for the non-elementary succinctness of RE($\neg$) versus automata.
† We are unsure about the tightness of the non-polynomial upper bounds in these columns.

$O(\ell)$. If the syntax is extended with counting operators, then each integer up to $N$ requires $O(\log N)$ bits to encode, and the resulting description length becomes $O(\ell \log N)$.

## 3. Learning Models

A concept class $\mathcal{C}$ is a series of collections of functions $\mathcal{C}_n \subseteq \{0,1\}^{\mathcal{X}_n}$, for $n \in \mathbb{N}$, where $\mathcal{X}_n = \{0,1\}^n$. Sometimes we abuse the notation and identify $\mathcal{C}$ with $\mathcal{C}_n$ where the meaning is clear from the context.

**PAC learning (Valiant, 1984).** A (possibly randomized) algorithm $A$ is said to *PAC learn* $\mathcal{C}_n$ if, for all $\epsilon, \delta \in (0,1)$, for all $n \in \mathbb{N}$, for all $c^\star \in \mathcal{C}_n$, and for all distributions $\mathcal{D}_n$ over $\mathcal{X}_n$, the following holds: Given access to i.i.d. examples $(x, c^\star(x))$ drawn from $\mathcal{D}_n$, the algorithm $A$ outputs a (description of) hypothesis $h$ such that, with probability at least $1-\delta$ (over the random draw of examples and the internal randomness of $A$), $\mathbb{P}_{x\sim\mathcal{D}_n}\left[h(x) \neq c^\star(x)\right] \leq \epsilon$.

We say that $\mathcal{C}_n$ is *tractably learnable* if there exists a learning algorithm $A$ and a polynomial $p$ such that, for every $n, \epsilon, \delta$, every distribution $\mathcal{D}_n$, and every $c^\star \in \mathcal{C}_n$, the algorithm $A$ PAC learns $\mathcal{C}_n$ and runs in time at most $p(n, 1/\epsilon, 1/\delta)$.

The learner may output an arbitrary hypothesis $h : \mathcal{X}_n \to \{0,1\}$ (not necessarily in $\mathcal{C}_n$), provided that $h$ can be evaluated in polynomial time in $(n, 1/\epsilon, 1/\delta)$. This setting is also known as improper learning (or representation-independent learning).

**Learning with membership queries (Angluin, 1987, 1988).** In the membership query (MQ) model, the learner is given the additional ability to actively request the label of any instance of its choice. Formally, besides receiving i.i.d. labeled examples $(x, c^\star(x))$ drawn from $\mathcal{D}_n$, the learner

may adaptively query an oracle on any $x \in \mathcal{X}_n$ to obtain $c^\star(x)$. A learning algorithm $A$ is said to *PAC+MQ learn* $\mathcal{C}_n$ if it satisfies the same accuracy and confidence guarantees as in the PAC model. Tractability is defined in the same way as for PAC learning.

We also consider $\gamma$-weak learnability, where the error of the hypothesis returned by the learning algorithm is just slightly better than a random guess, that is, $\mathbb{P}_{x\sim\mathcal{D}_n}\left[h(x) \neq c^\star(x)\right] \leq 1/2 - \gamma$, where $\gamma > 0$ is either a constant or $\frac{1}{\text{poly}(n)}$. Computational hardness results for improper learning in a distribution-free setting are translated to the hardness of improper weak learning, since, by using boosting algorithms (Schapire, 1990; Freund, 1995; Schapire and Freund, 2013), if there is an efficient algorithm with error at most $\frac{1}{2} - \frac{1}{\text{poly}(n)}$, then there is an efficient boosting algorithm that learns $\mathcal{C}_n$. When we consider hardness of weak learning without specifying $\gamma$ we mean that there is no tractable $\gamma$-weak learning algorithm for any inverse-polynomial $\gamma$.

## 4. Hardness of PAC Learning

In this section, we prove the hardness of PAC learning REs under several standard assumptions, both under arbitrary input distributions and under the uniform distribution.

### 4.1. Distribution-free hardness

We start with a distribution-free hardness result, obtained via an efficient prediction-preserving reduction from PAC learning Disjunctive Normal Forms (DNFs) to learning regular expressions. Then, known improper hardness results for DNFs in the PAC model directly imply the improper hardness of learning regular expressions.

We define DNFs as follows. Fix a set of Boolean variables $X = \{x_1, \ldots, x_n\}$. A literal is either a variable $x_i$ or its negation $\neg x_i$. A term (or clause) is a conjunction of one or more distinct literals, $T = \ell_{i_1} \wedge \ell_{i_2} \wedge \cdots \wedge \ell_{i_k}$, where each $\ell_{i_j}$ is a literal over $X$ and no variable appears twice in $T$. The width of $T$ is $k$, and can be at most $n$. A DNF formula is a disjunction of terms, $\Phi = T_1 \vee T_2 \vee \cdots \vee T_m$, where $m$ is a function of $n$. The size of $\Phi$ is defined as its total number of literals, $|\Phi| = \sum_{j=1}^{m} |T_j|$, with $|T_j|$ equal to the width of $T_j$. For an assignment $\boldsymbol{x} \in \{0,1\}^n$, the value $\Phi(\boldsymbol{x}) \in \{0,1\}$ is obtained under the usual Boolean semantics.

**Lemma 4.1 (Polynomial-Size DNF $\to$ RE Translation)** *Let $\Phi : \{0,1\}^n \to \{0,1\}$ be a DNF with $m(n)$ terms. There is a plain regular expression $R_\Phi$ over $\{0,1\}$ of size $O(mn)$ such that for every $\boldsymbol{x} \in \{0,1\}^n$, $\boldsymbol{x} \in L(R_\Phi) \leftrightarrow \Phi(\boldsymbol{x}) = 1$.*

**Proof** Write $\Phi = \bigvee_{j=1}^{m} T_j$, where each term $T_j$ is a conjunction of literals over $\{x_1, \ldots, x_n\}$. For a given term $T_j$, define a regular expression $R_{T_j} = \gamma_1^{(j)}\gamma_2^{(j)}\cdots\gamma_n^{(j)}$, where for each position $i$,

$$\gamma_i^{(j)} = \begin{cases} 1 & \text{if the literal } x_i \text{ occurs in } T_j, \\ 0 & \text{if the literal } \neg x_i \text{ occurs in } T_j, \\ (0 \vee 1) & \text{if } x_i \text{ does not occur in } T_j. \end{cases}$$

Set $R_\Phi = \left(R_{T_1} \vee \cdots \vee R_{T_{m(n)}}\right)$. By construction, $R_{T_j}$ matches exactly those length-$n$ strings that satisfy the constraints of $T_j$, and $R_\Phi$ matches a string iff at least one term holds, i.e., iff $\Phi(\boldsymbol{x}) = 1$. The size is $O(n)$ per term, so $|R_\Phi| = O(mn)$.

For example, let $\Phi = (x_1 \wedge \neg x_3 \wedge x_4) \vee (\neg x_2 \wedge x_3)$ be a DNF over variables $\{x_1, x_2, x_3, x_4\}$. The Equivalent RE would be $R_\phi = (1(0 \vee 1)01) \vee ((0 \vee 1)01(0 \vee 1))$. ■

The above lemma shows that poly-sized DNFs can be expressed by poly-sized REs. Hence, hardness of learning DNFs implies hardness of learning REs. Thus, we establish the hardness of learning REs via known results on DNFs.

Distribution-free hardness of PAC learning DNFs was established under the random $K$-SAT assumption (Daniely and Shalev-Shwartz, 2016). Informally, this assumption states that no polynomial-time algorithm can efficiently refute (certify unsatisfiability of) typical random $K$-SAT formulas. Under the local pseudorandom generator (local PRG) assumption, tighter distribution-free hardness and distribution-specific hardness for a product distribution (distinct from the uniform distribution) were established in Daniely and Vardi (2021). See Section 4.2 for the formal definition of local PRGs, where we use it explicitly to show hardness of learning REs under the uniform distribution. Finally, under a variant of the sparse learning parity with noise assumption, Applebaum et al. (2025) proved distribution-free hardness of learning DNFs. Informally, this assumption states that even when each linear equation involves only a few variables, it is still computationally hard to recover the hidden parity in the presence of random noise.

**Theorem 4.2 (Distribution-Free Hardness of PAC Learning REs)** *Assuming either the local-PRG assumption (Assumption 4.3), the random $K$-SAT assumption (Daniely and Shalev-Shwartz, 2016), or the sparse learning-parity-with-noise assumption from Applebaum et al. (2025), for any constant $\epsilon > 0$, there is no tractable algorithm that weakly PAC learns the concept class*

$$\mathcal{C}_n = \left\{ c : \{0,1\}^n \to \{0,1\} \;\middle|\; c \text{ is representable by a regular expression of size} \leq n^\epsilon \right\}.$$

**Proof** Under the local-PRG assumption (Daniely and Vardi, 2021), or the hardness assumption for sparse learning parity with noise (Applebaum et al., 2025), weakly learning DNFs with $m(n) = \omega(1)$ terms is hard. Take $m(n) = \log(n)$, then by Lemma 4.1, it is hard to weakly learn REs of size $O(n \log(n))$. By a standard scaling argument, it implies that for any constant $\epsilon > 0$, it is hard to learn REs of size $n^\epsilon$. Indeed, define $\tilde{n} = n^{1+1/\epsilon}$, and note that $\tilde{n}^\epsilon = n^{1+\epsilon}$ is larger than $O(n \log(n))$ for a sufficiently large $n$. For an example $(\boldsymbol{x}, y) \in \{0,1\}^n \times \{0,1\}$ realizable by an RE of size $O(n \log(n))$, by padding the input $\boldsymbol{x}$ with zeros, we obtain $\tilde{\boldsymbol{x}} \in \{0,1\}^{\tilde{n}}$ where $(\tilde{\boldsymbol{x}}, y)$ is realizable by an RE of size at most $\tilde{n}^\epsilon$. Thus, it is hard to weakly learn REs over $\{0,1\}^{\tilde{n}}$ of size $\tilde{n}^\epsilon$.

Under the $K$-SAT assumption, weakly learning DNFs with $m(n) = \omega(\log(n))$ terms is hard (Daniely and Shalev-Shwartz, 2016). Take $m(n) = \log^2(n)$, then by Lemma 4.1, it is hard to learn REs of size $O(n \log^2(n))$. By a similar scaling argument, we get that it is hard to weakly learn REs of size $n^\epsilon$. ■

### 4.2. Hardness under the uniform distribution

For the uniform distribution, the DNF route is insufficient: it is open whether DNFs are tractably learnable under the uniform distribution, and the best known algorithms are quasi-polynomial (Verbeurgt, 1990; Linial et al., 1993).

Therefore, we follow the approach introduced by Daniely and Vardi (2021), showing that if regular expressions could be weakly learned tractably, then one could distinguish random strings from pseudorandom strings generated by a local pseudorandom generator. Their framework was

used to prove distribution-free and distribution-specific hardness of PAC learning for several important concept classes, including DFAs. However, for our purpose, we need a construction tailored specifically to regular expressions.

**Local pseudorandom generators (PRGs).** An $(n, m, k)$-hypergraph is a hypergraph over vertex set $[n]$ with $m$ (ordered) hyperedges $E_1, \ldots, E_m$, each of cardinality $k$, where all vertices in a hyperedge are distinct. We denote by $\mathcal{G}_{n,m,k}$ the distribution obtained by selecting each hyperedge independently and uniformly from all $n \cdot (n-1) \cdots (n-k+1)$ possible ordered $k$-tuples.

Let $P : \{0,1\}^k \to \{0,1\}$ be a predicate, and let $G$ be an $(n, m, k)$-hypergraph. Goldreich's pseudorandom generator (PRG) (Goldreich, 2000) is defined as

$$f_{P,G} : \{0,1\}^n \to \{0,1\}^m, \qquad f_{P,G}(\boldsymbol{x}) = \big(P(\boldsymbol{x}|_{E_1}), \ldots, P(\boldsymbol{x}|_{E_m})\big).$$

The integer $k$ is called the *locality* of the PRG. If $k$ is a constant, the PRG is said to be *local*. The PRG has *polynomial stretch* if $m = n^s$ for some constant $s > 1$. We denote by $\mathcal{F}_{P,n,m}$ the family of functions $f_{P,G}$ where $G$ ranges over $(n, m, k)$-hypergraphs. Sampling from $\mathcal{F}_{P,n,m}$ means choosing $G \sim \mathcal{G}_{n,m,k}$. We write $G \xleftarrow{R} \mathcal{G}_{n,m,k}$ for a random choice of $G$, and $\boldsymbol{x} \xleftarrow{R} \{0,1\}^n$ for a uniformly random string.

The family $\mathcal{F}_{P,n,m}$ is an $\varepsilon$-pseudorandom generator (PRG) if for every probabilistic polynomial-time algorithm $A$, the distinguishing advantage

$$\left| \mathbb{P}_{G \xleftarrow{R} \mathcal{G}_{n,m,k}, \boldsymbol{x} \xleftarrow{R} \{0,1\}^n} \big[A(G, f_{P,G}(\boldsymbol{x})) = 1\big] - \mathbb{P}_{G \xleftarrow{R} \mathcal{G}_{n,m,k}, \boldsymbol{y} \xleftarrow{R} \{0,1\}^m} \big[A(G, \boldsymbol{y}) = 1\big] \right|$$

is at most $\varepsilon$.

**Assumption 4.3 (Local PRG Assumption)** *For every constant $s > 1$, there exists a constant $k$ and a predicate $P : \{0,1\}^k \to \{0,1\}$ such that $\mathcal{F}_{P,n,n^s}$ is a $\frac{1}{3}$-PRG.*

Requiring a constant distinguishing advantage is weaker than the more common requirement of negligible advantage (see, e.g., Applebaum (2016); Applebaum and Lovett (2016); Couteau et al. (2018)). Local PRGs with polynomial stretch have been extensively studied and applied, for example in secure computation with constant overhead and in general-purpose obfuscation via constant-degree multilinear maps (Ishai et al., 2008; Applebaum et al., 2017; Lin, 2016; Lin and Vaikuntanathan, 2016). A key theoretical foundation was provided by Applebaum (2013), who showed that the above assumption holds if there exists a sensitive local predicate $P$ such that $\mathcal{F}_{P,n,n^s}$ is one-way, a variant of Goldreich's original one-wayness assumption (Goldreich, 2000). Assumption 4.3 was used by Daniely and Vardi (2021); Daniely et al. (2023) for proving hardness of learning of various classes.

**Theorem 4.4 (Hardness of PAC Learning REs under the Uniform Distribution)** *Under the local-PRG assumption (Assumption 4.3), for any constants $\epsilon, \gamma > 0$, there is no tractable algorithm that $\gamma$-weakly learns the concept class*

$$\mathcal{C}_n = \big\{c : \{0,1\}^n \to \{0,1\} \;\big|\; c \text{ is representable by a regular expression of size} \leq n^\epsilon\big\}$$

*on the uniform distribution over $\{0,1\}^n$.*

We note that the hardness result holds even for regular expressions using only union and concatenation (i.e., without the Kleene star), provided that we bound the RE size by $O(n)$ (instead of $n^\epsilon$).

The formal proof is rather technical and appears in Appendix A. The high-level strategy is as follows. For a hyperedge $E = (i_1, \ldots, i_k)$, we denote $\boldsymbol{x}|_E = (x_{i_1}, \ldots, x_{i_k})$. Let $s > 1$. Given a sequence $(E_1, y_1), \ldots, (E_{n^s}, y_{n^s})$, where $E_1, \ldots, E_{n^s}$ are i.i.d. random hyperedges, we aim to design an algorithm that distinguishes whether $\boldsymbol{y} = (y_1, \ldots, y_{n^s})$ is uniformly random or pseudorandom. Specifically, in the pseudorandom case we set $\boldsymbol{y} = \big(P(\boldsymbol{x}|_{E_1}), \ldots, P(\boldsymbol{x}|_{E_{n^s}})\big)$, where $\boldsymbol{x} \in \{0,1\}^n$ is chosen uniformly at random. Assuming the existence of a tractable weak learning algorithm for regular expressions for the uniform distribution, we construct a training set that is realizable by regular expressions when $\boldsymbol{y}$ is pseudorandom. The hypothesis returned by the weak learner then acts as a distinguisher between the random and pseudorandom cases, thereby violating the PRG assumption.

### 4.3. Comparison to Angluin et al. (2013); Chen (2014)

The learnability of a subclass of REs called *shuffle ideals* was investigated by Angluin et al. (2013). They established tractable PAC learnability under the uniform distribution, and, under cryptographic assumptions, hardness of improper PAC learning for a sufficiently large (non-binary) alphabets. This hardness does not carry over when membership queries are allowed. They further noted that extending this hardness result to smaller alphabets, in particular the binary alphabet, remains an open question. Subsequently, Chen (2014) proved tractable learnability of this class for specific distributions in the statistical query model.

In contrast, we study plain regular expressions over the binary alphabet. We first establish distribution-free hardness in the PAC model. We then prove hardness under the uniform distribution, in contrast to the positive result for the subclass of shuffle ideals. Finally, in the next section, we show both distribution-free and distribution-specific hardness even in the presence of membership queries, a setting for which no such hardness results were previously known.

We note that the construction of Angluin et al. (2013) can be modified to yield distribution-free hardness (under cryptographic assumptions) for learning general regular expressions. This can be done by converting their strings over a non-binary alphabet into binary strings, incurring only a logarithmic overhead in size. The resulting regular expressions remain valid, since we do not restrict ourselves to the subclass of shuffle ideals. This observation was not mentioned by Angluin et al. (2013), who did not ask about standard REs and left the question of binary alphabets explicitly open.

## 5. Hardness of MQ Learning

In this section, we study the setting where the learner is allowed to issue membership queries, in addition to sampling random labeled examples from the distribution.

### 5.1. Distribution-free hardness

The hardness of learning DNFs, discussed in Section 4.1, extends to this setting as well. In particular, Angluin and Kharitonov (1995) showed that if there exists a non-uniform one-way function that cannot be inverted by polynomial-size circuits, then the hardness of distribution-free PAC learning

DNFs carries over to the model with membership queries, i.e., DNFs are either efficiently PAC-learnable from random examples alone, or else not even efficiently learnable with queries. Hence, using the same arguments from Section 4.1, we have the following:

**Theorem 5.1 (Distribution-Free Hardness of PAC+MQ Learning REs)** *Assume there exists a non-uniform one-way function, and moreover assume either the local-PRG assumption (Assumption 4.3), the random $K$-SAT assumption (Daniely and Shalev-Shwartz, 2016), or the sparse learning-parity-with-noise assumption from Applebaum et al. (2025). Then, for any constant $\epsilon > 0$, there is no tractable algorithm that PAC+MQ learns the concept class*

$$\mathcal{C}_n = \left\{c : \{0,1\}^n \to \{0,1\} \;\middle|\; c \text{ is representable by a regular expression of size} \leq n^\epsilon\right\}.$$

### 5.2. Distribution-specific hardness of learning extended REs

So far we have established distribution-free hardness of learning regular expressions, even when membership queries are allowed. Moreover, we established hardness of learning regular expressions without queries under the uniform distribution. In this section, we show hardness of learning with queries under the uniform distribution, provided that regular expressions are extended with complement or intersection. Our approach is to reduce PAC+MQ learning Boolean formulae to the problem of learning such extended REs, from which the hardness result follows by Kharitonov (1993).

**Lemma 5.2 (Polynomial-Size Boolean Formula $\to$ RE($\neg$) / RE($\cap$) Translation)** *Let $\Sigma = \{0,1\}$ and let $\varphi(x_1, \ldots, x_n)$ be a Boolean formula of size $|\varphi|$. There exists a regular expression $R_\varphi$ over $\Sigma$ using only union and concatenation together with either complement (for RE($\neg$)) or intersection (for RE($\cap$)), such that $L(R_\varphi) \cap \{0,1\}^n = \left\{(a_1, \ldots, a_n) \in \{0,1\}^n : \varphi(a_1, \ldots, a_n) = 1\right\}$. Moreover, $|R_\varphi| = O(|\varphi| n)$, and $|R_\varphi| = O(|\varphi| \log n)$ if the counting operator is allowed.*

**Proof** We start with the case of RE($\cap$): First convert $\varphi$ to negation normal form (NNF), pushing all negations to literals. This increases the size by at most a constant factor. Define $R(\cdot)$ by structural induction on the NNF structure and set $R_\varphi := R(\varphi)$. For literals ($i \in [n]$),

$$R(x_i) := (0 \vee 1)^{i-1} 1 (0 \vee 1)^{n-i}, \qquad R(\neg x_i) := (0 \vee 1)^{i-1} 0 (0 \vee 1)^{n-i}.$$

For connectives,

$$R(\alpha \vee \beta) := R(\alpha) \vee R(\beta), \qquad R(\alpha \wedge \beta) := R(\alpha) \wedge R(\beta).$$

By induction on the NNF structure we show

$$L\big(R(\varphi)\big) \cap \{0,1\}^n = \{\mathbf{a} \in \{0,1\}^n : \varphi(\mathbf{a}) = 1\}.$$

The base cases hold by construction of $R(x_i)$ and $R(\neg x_i)$. For the inductive steps, use $L(R(\alpha \vee \beta)) = L(R(\alpha)) \cup L(R(\beta))$ and $L(R(\alpha \wedge \beta)) = L(R(\alpha)) \cap L(R(\beta))$. The claim follows from the induction hypothesis.

**The size of the regular expression.** Each literal contributes $O(n)$ symbols (or $O(\log n)$ with the counting operator), and each connective adds $O(1)$. The NNF conversion is linear-size, so $|R_\varphi| = O(|\varphi| n)$, or $O(|\varphi| \log n)$ with counting.

**Conclusion for** ($\mathsf{RE}(\neg)$)**.** Since intersection is definable from complement and union by De Morgan, with constant overhead, $R(\alpha \wedge \beta) := \neg\big(\neg R(\alpha) \vee \neg R(\beta)\big)$, the above construction immediately yields an $\mathsf{RE}(\neg)$ expression of the same asymptotic size. Thus, the correctness and size bounds carry over to $\mathsf{RE}(\neg)$ as well. ■

The above lemma shows that polynomial-size Boolean formulas can be represented by polynomial-size REs extended with intersection or with negation. Hence, hardness of learning Boolean formulas implies hardness of learning $\mathsf{RE}(\cap)$ and $\mathsf{RE}(\neg)$. Thus, we establish the hardness of learning extended REs via known results on Boolean formulas (Kharitonov, 1993).

**Theorem 5.3 (Hardness of PAC+MQ Learning $\mathsf{RE}(\cap)$ or $\mathsf{RE}(\neg)$ Under the Uniform Distribution)** *Assuming one of the cryptographic assumptions: RSA, factoring Blum integers, or quadratic residues (Kharitonov, 1993), for any constant $\epsilon > 0$, there is no tractable algorithm that PAC+MQ learns the concept class*

$$\mathcal{C}_n = \big\{c : \{0,1\}^n \to \{0,1\} \;\big|\; c \text{ is representable by a RE with intersection or complement of size} \leq n^\epsilon\big\},$$

*on the uniform distribution over $\{0,1\}^n$.*

**Proof** Under one of the assumptions: RSA, factoring Blum integers, or quadratic residues, there exists some constant $s > 1$ such that Boolean formulas of size $n^s$ cannot be tractably weakly learned with membership queries on the uniform distribution over $\{0,1\}^n$ (Kharitonov, 1993). By Lemma 5.2, this implies hardness of weakly learning $\mathsf{RE}(\cap)$ or $\mathsf{RE}(\neg)$ of size $n^{s+1}$. By a standard scaling argument, similar to the proof of Theorem 4.2, we conclude that for any $\epsilon > 0$, it is hard to weakly learn such REs of size $n^\epsilon$. ■

## 6. Discussion and Open Questions

Regular expressions are a fundamental object in computer science, yet the computational complexity of learning them has surprisingly not been previously rigorously established. In this work, we close this imporant gap in the literature by proving hardness results both in the PAC model and in the membership query setting, under distribution-free learning as well as under the uniform distribution. Beyond the fact that we now have a reference to rely on regarding this hardness, closing this gap is also important since it reveals subtleties about the tractability of learning regular expressions, in particular under membership queries.

As we have discussed, learning REs is not the same as learning DFAs or NFAs, even though all three characterize the class of regular languages. The key distinction lies in the measure of complexity, namely, the description length, which as shown in Section 2.1, can differ substantially across these representations. This difference has concrete consequences for learnability: DFAs are efficiently learnable with membership queries, whereas we prove that REs remain hard in the same model.

A key takeaway is that what truly matters is the complexity measure (or description length, or equivalently 'prior') induced by the model, rather than the concept class itself. Although REs, DFAs, and NFAs all define the same family of regular languages and are thus equivalent in a uniform sense, this equivalence is largely irrelevant from a learning perspective. For any finite $n$, every predictor

defines a finite, and therefore regular language. The real question is the quantitative *complexity measure* (e.g., DFA size or RE length), and these differ dramatically between different "equivalent" models. Here, we focused only on polynomial hardness (a fairly crude notion), but in a more refined analysis, e.g., of sample complexity or exact runtime, finer-grained quantitative differences become even more important.

As a direction for future work, we point out that to the best of our knowledge, it is still unknown whether there exists a tractable algorithm for learning plain REs or NFAs with membership queries under the uniform distribution. Also, while REs can be efficiently converted into NFAs, NFAs may be exponentially more succinct than REs. However, it remains unclear whether this asymmetry implies a genuine separation in their learnability.

## Acknowledgments

We thank Aryeh Kontorovich and Dana Angluin for helpful discussions.

This work was conducted as part of the NSF-supported Institute for Data, Econometrics, Algorithms and Learning (IDEAL), under grants NSF ECCS-2217023 and NSF EECS-2216899. GV is supported by the Israel Science Foundation (grant No. 2574/25), by a research grant from Mortimer Zuckerman (the Zuckerman STEM Leadership Program), and by research grants from the Center for New Scientists at the Weizmann Institute of Science, and the Shimon and Golde Picker – Weizmann Annual Grant.

## References

Dana Angluin. Learning regular sets from queries and counterexamples. *Information and computation*, 75(2):87–106, 1987.

Dana Angluin. Queries and concept learning. *Machine learning*, 2(4):319–342, 1988.

Dana Angluin and Michael Kharitonov. When won't membership queries help? *Journal of Computer and System Sciences*, 50(2):336–355, 1995.

Dana Angluin, James Aspnes, Sarah Eisenstat, and Aryeh Kontorovich. On the learnability of shuffle ideals. *The Journal of Machine Learning Research*, 14(1):1513–1531, 2013.

Benny Applebaum. Cryptographic hardness of random local functions—survey. *Computational Complexity*, 22:599–650, 2013.

Benny Applebaum. Cryptography in nc0. *SIAM Journal on Computing*, 45(4):903–953, 2016.

Benny Applebaum and Shachar Lovett. Pseudorandomness in $NC^0$ and cryptography. In *CCC*, 2016.

Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Low-complexity cryptographic primitives. In *ICALP*, 2017.

Benny Applebaum, Dung Bui, Geoffroy Couteau, and Nikolas Melissaris. Structured-seed local pseudorandom generators and their applications. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2025)*, pages 63–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2025.

Dongqu Chen. Learning shuffle ideals under restricted distributions. *Advances in Neural Information Processing Systems*, 27, 2014.

Geoffroy Couteau et al. Pseudorandomness in cryptography. In *ACM CCS*, 2018.

Zui Ruan Dang. On the complexity of a finite automaton corresponding to a generalized regular expression. In *Doklady Akademii Nauk*, volume 213, pages 26–29. Russian Academy of Sciences, 1973.

Amit Daniely and Shai Shalev-Shwartz. Complexity theoretic limitations on learning dnf's. In *Conference on Learning Theory*, pages 815–830. PMLR, 2016.

Amit Daniely and Gal Vardi. From local pseudorandom generators to hardness of learning. In *Conference on Learning Theory*, pages 1358–1394. PMLR, 2021.

Amit Daniely, Nati Linial, and Shai Shalev-Shwartz. From average case complexity to improper learning complexity. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 441–448, 2014.

Amit Daniely, Nati Srebro, and Gal Vardi. Computational complexity of learning neural networks: Smoothness and degeneracy. *Advances in Neural Information Processing Systems*, 36:76272–76297, 2023.

Andrzej Ehrenfeucht and Paul Zeiger. Complexity measures for regular expressions. *Journal of Computer and System Sciences*, 12(2):134–146, 1976.

Keith Ellul, Bryan Krawetz, Jeffrey Shallit, and Ming-Wei Wang. Regular expressions: New results and open problems. *J. Autom. Lang. Comb.*, 10(4):407–437, 2005.

Yoav Freund. Boosting a weak learning algorithm by majority. *Information and computation*, 121 (2):256–285, 1995.

Wouter Gelade. Succinctness of regular expressions with interleaving, intersection and counting. *Theoretical Computer Science*, 411(31-33):2987–2998, 2010.

Wouter Gelade and Frank Neven. Succinctness of the complement and intersection of regular expressions. *ACM Transactions on Computational Logic (TOCL)*, 13(1):1–19, 2012.

Wouter Gelade, Marc Gyssens, and Wim Martens. Regular expressions with counting: weak versus strong determinism. *SIAM Journal on Computing*, 41(1):160–190, 2012.

E Mark Gold. Complexity of automaton identification from given data. *Information and control*, 37(3):302–320, 1978.

Oded Goldreich. Candidate one-way functions based on expander graphs. *Electronic Colloquium on Computational Complexity (ECCC)*, 7(090), 2000.

Hermann Gruber and Markus Holzer. From finite automata to regular expressions and back—a summary on descriptional complexity. *International Journal of Foundations of Computer Science*, 26 (08):1009–1040, 2015.

Andrew Hume. A tale of two greps. *Software: Practice and Experience*, 18(11):1063–1072, 1988.

Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, Amit Sahai, and David Wagner. Secure arithmetic computation with constant computational overhead. In *CRYPTO*, 2008.

Michael Kearns and Leslie Valiant. Cryptographic limitations on learning boolean formulae and finite automata. *Journal of the ACM (JACM)*, 41(1):67–95, 1994.

Michael Kharitonov. Cryptographic hardness of distribution-specific learning. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 372–381, 1993.

Pekka Kilpeläinen and Rauno Tuhkanen. Regular expressions with numerical occurrence indicators-preliminary results. *SPLST*, 3:163–173, 2003.

Stephen Cole Kleene. *Representation of events in nerve nets and finite automata*, volume 34. Princeton University Press Princeton, 1956.

Huijia Lin. Indistinguishability obfuscation from constant-degree graded encodings. In *FOCS*, 2016.

Huijia Lin and Vinod Vaikuntanathan. Generalized multilinear maps from obfuscation. In *FOCS*, 2016.

Nathan Linial, Yishay Mansour, and Noam Nisan. Constant depth circuits, fourier transform, and learnability. *Journal of the ACM (JACM)*, 40(3):607–620, 1993.

Albert R Meyer and Michael J Fischer. Economy of description by automata, grammars, and formal systems. In *12th annual symposium on switching and automata theory (swat 1971)*, pages 188–191. IEEE Computer Society, 1971.

Albert R Meyer and Larry J Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *SWAT*, volume 72, pages 125–129, 1972.

Frank R Moore. On the bounds for state-set size in the proofs of equivalence between deterministic, nondeterministic, and two-way finite automata. *IEEE Transactions on computers*, 100(10):1211–1214, 1971.

Leonard Pitt and Manfred K Warmuth. Prediction-preserving reducibility. *Journal of Computer and System Sciences*, 41(3):430–467, 1990.

Leonard Pitt and Manfred K Warmuth. The minimum consistent dfa problem cannot be approximated within any polynomial. *Journal of the ACM (JACM)*, 40(1):95–142, 1993.

Michael O Rabin and Dana Scott. Finite automata and their decision problems. *IBM journal of research and development*, 3(2):114–125, 1959.

Robert E Schapire. The strength of weak learnability. *Machine learning*, 5(2):197–227, 1990.

Robert E Schapire and Yoav Freund. Boosting: Foundations and algorithms. *Kybernetes*, 42(1): 164–166, 2013.

C. M. Sperberg-McQueen and H. Thompson. Xml schema, 2005. URL https://www.w3.org/XML/.

LJ Stockmeyer and AR Meyer. Word problems requiring exponential time: preliminary report, in5th symp. on theory of computing'. 1973.

Lena Strobl, William Merrill, Gail Weiss, David Chiang, and Dana Angluin. What formal languages can transformers express? a survey. *Transactions of the Association for Computational Linguistics*, 12:543–561, 2024.

Ken Thompson. Programming techniques: Regular expression search algorithm. *Communications of the ACM*, 11(6):419–422, 1968.

Leslie G Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

Karsten Verbeurgt. Learning dnf under the uniform distribution in quasi-polynomial time. In *Proceedings of the third annual workshop on Computational learning theory*, pages 314–326, 1990.

Larry Wall, Tom Christiansen, and Randal L Schwartz. Programming perl. 1999.

## Appendix A. Hardness of Regular Expressions in PAC Learning under the Uniform Distribution

**Proof** [of Theorem 4.4] For $\boldsymbol{x} \in \{0,1\}^n$ and coordinates $(i_1, \ldots, i_\ell)$ we write $\boldsymbol{x}|_{(i_1,\ldots,i_\ell)} = (x_{i_1}, \ldots, x_{i_\ell})$, that is, the restriction of $\boldsymbol{x}$ to these coordinates. In particular, for a hyperedge $E = (i_1, \ldots, i_k)$, we write $\boldsymbol{x}|_E = (x_{i_1}, \ldots, x_{i_k})$.

Our high-level strategy is as follows. Let $s > 1$. Given a sequence $(E_1, y_1), \ldots, (E_{n^s}, y_{n^s})$, where $E_1, \ldots, E_{n^s}$ are i.i.d. random hyperedges, we aim to design an algorithm that distinguishes whether $\boldsymbol{y} = (y_1, \ldots, y_{n^s})$ is uniformly random or pseudorandom. Specifically, in the pseudorandom case we set $\boldsymbol{y} = \big(P(\boldsymbol{x}|_{E_1}), \ldots, P(\boldsymbol{x}|_{E_{n^s}})\big)$, where $\boldsymbol{x} \sim \{0,1\}^n$ is chosen uniformly at random. Assuming the existence of an efficient weak learning algorithm for regular expressions for the uniform distribution, we construct a training set that is realizable by regular expressions when $\boldsymbol{y}$ is pseudorandom. The hypothesis returned by the weak learner then acts as a distinguisher between the random and pseudorandom cases, thereby violating the PRG assumption.

Let $\mathcal{D}$ be the uniform distribution on $\{0,1\}^{n^\alpha}$, where $\alpha \geq 1$ (to be determined at the end of the proof). Suppose for contradiction that there exists an efficient algorithm $L$ that learns regular expressions of size $n \log^2(n)$ under $\mathcal{D}$. Let $p(n)$ be a polynomial such that $L$ uses at most $p(n)$ samples and, with probability at least $\frac{3}{4}$, outputs a hypothesis $h$ with error at most $\frac{1}{2} - \gamma$.

Choose a constant $s > 1$ such that $n^s \geq p(n) + n$ for all sufficiently large $n$. By Assumption 4.3, for every $s$, there exists a constant $k$ and a predicate $P : \{0,1\}^k \to \{0,1\}$ such that $\mathcal{F}_{P,n,n^s}$ is a $\frac{1}{3}$-PRG. We design an algorithm $A$ that distinguishes random from pseudorandom with advantage greater than $\frac{1}{3}$, contradicting the assumption. The algorithm $A$ uses $p(n)$ samples to simulate the learning algorithm $L$, along with an additional held-out set of $n$ samples that are not observed by $L$.

**Hyperedge encodings.** We encode a hyperedge $E = (i_1, \ldots, i_k)$ by a vector $\boldsymbol{z}^E \in \{0,1\}^{kn}$, defined as the concatenation of $k$ vectors in $\{0,1\}^n$, where the $j$-th vector has a zero in the $i_j$-th coordinate and ones in all other coordinates (similar to a one-hot encoding, except that the marked entry is zero and all others are one). This construction uniquely represents the hyperedge $E$. Let $\tilde{\boldsymbol{z}}^E \in \{0,1\}^{k\log(n)}$ denote the *compressed encoding* of $E$, defined as the concatenation of the binary representations (each of length $\log(n)$) of the $k$ vertices of $E$. We say that $\tilde{\boldsymbol{z}} \in \{0,1\}^{n^\alpha}$ is an *extended compressed encoding* of $E$ if $\tilde{\boldsymbol{z}}|_{(1,\ldots,k\log(n))} = \tilde{\boldsymbol{z}}^E|_{(1,\ldots,k\log(n))}$, that is, the first $k\log(n)$ coordinates of $\tilde{\boldsymbol{z}}$ equal $\tilde{\boldsymbol{z}}^E$.

If $\tilde{\boldsymbol{z}}$ is uniform in $\{0,1\}^{n^\alpha}$, then the probability that it is a valid extended compressed encoding, namely, that each two of the first $k$ blocks of size $\log(n)$ encode different indices, is simply the probability that $k$ independently chosen indices are all distinct,

$$\begin{aligned}\mathbb{P}[\tilde{\boldsymbol{z}} \text{ encodes a hyperedge}] &= \frac{n\cdot(n-1)\cdots(n-k+1)}{n^k} \\ &\geq \left(1-\tfrac{k}{n}\right)^k \\ &\geq 1-\tfrac{\gamma}{2},\end{aligned} \tag{1}$$

for sufficiently large $n$. Note that we need the compressed encoding exactly for (1): if we used $k$ blocks of size $n$, then each block would have to be a one-hot vector, which occurs with negligible probability.

**Simulating examples for $L$ (efficient weak learner for RE under the uniform distribution).** For $\boldsymbol{x} \in \{0,1\}^n$ , let $P_{\boldsymbol{x}} : \{0,1\}^{kn} \to \{0,1\}$ and $\tilde{P}_{\boldsymbol{x}} : \{0,1\}^{k\log(n)} \to \{0,1\}$ be such that for every hyperedge $E$, it holds that $P_{\boldsymbol{x}}(\boldsymbol{z}^E) = \tilde{P}_{\boldsymbol{x}}(\tilde{\boldsymbol{z}}^E) = P(\boldsymbol{x}|_E)$.

Given $(E_1, y_1), \ldots, (E_{n^s}, y_{n^s})$ with $E_i$ random hyperedges, algorithm $A$ must distinguish whether $\boldsymbol{y} = (y_1, \ldots, y_{n^s})$ is uniformly random or pseudorandom. In the pseudorandom case we have $\boldsymbol{y} = \big(P(\boldsymbol{x}|_{E_1}), \ldots, P(\boldsymbol{x}|_{E_{n^s}})\big) = \big(\tilde{P}_{\boldsymbol{x}}(\tilde{\boldsymbol{z}}^{E_1}), \ldots, \tilde{P}_{\boldsymbol{x}}(\tilde{\boldsymbol{z}}^{E_{n^s}})\big)$ for some uniformly random $\boldsymbol{x} \in \{0,1\}^n$. Denote by $\boldsymbol{E} = (\tilde{\boldsymbol{z}}^{E_1}, y_1), \ldots, (\tilde{\boldsymbol{z}}^{E_{n^s}}, y_{n^s})$ the corresponding encoded hyperedges sample.

Algorithm $A$ runs $L$ on oracle access to random examples drawn from $\mathcal{D}$, the uniform distribution on $\{0,1\}^{n^\alpha}$. For each $i \in [n^s]$, the oracle samples $\tilde{\boldsymbol{z}}_i \sim \mathcal{D}$.

- If $\tilde{\boldsymbol{z}}_i$ is not an extended compressed encoding (this happens with probability at most $\gamma/2$ by Eq. (1)), return $(\boldsymbol{z}'_i, y'_i)$ with $\boldsymbol{z}'_i = \tilde{\boldsymbol{z}}_i$, $y'_i = 1$.
- Otherwise, replace the first $k\log(n)$ coordinates of $\tilde{\boldsymbol{z}}_i$ with $\tilde{\boldsymbol{z}}^{E_i}$, and let $\boldsymbol{z}'_i$ denote the resulting string. Output $(\boldsymbol{z}'_i, y'_i)$, where $y'_i = y_i$. This modification preserves the uniform distribution $\boldsymbol{z}'_i \sim \mathcal{D}$, since a random hyperedge is replaced by another random hyperedge.

Denote $S_t = \{(\boldsymbol{z}'_i, y'_i) : i \in [p(n)]\}$ and $S_v = \{(\boldsymbol{z}'_i, y'_i) : i \in \{p(n)+1, \ldots, p(n)+n\}\}$. Recall that $n^s \geq p(n) + n$, and that algorithm $L$ observes only $p(n)$ samples. Thus, $S_t$ serves as the training set for $L$, while $S_v$ acts as a validation set.

**Distinguisher algorithm $A$.** Let $h$ be the hypothesis produced by $L$ given $S_t$. Denote the error of $h$ on $S_v$ by $\widehat{\mathtt{Err}}_{S_v}(h) = \frac{1}{n}\sum_{(\boldsymbol{z}'_i, y'_i)\in S_v} \mathbb{I}\{h(\boldsymbol{z}'_i) \neq y'_i\}$.

- If $\widehat{\mathtt{Err}}_{S_v}(h) \leq \frac{1}{2} - \frac{\gamma}{2}$ then $A$ returns 1 (pseudorandom).
- Otherwise $A$ returns 0 (random).

We show below that if $\boldsymbol{E}$ is pseudorandom, then $A$ returns 1 with probability greater than $2/3$, and if $\boldsymbol{E}$ is random, then $A$ returns 0 with probability greater than $2/3$.

**Regular expression construction.** We construct a regular expression $R$ such that, if $\boldsymbol{y}$ is pseudorandom, then all examples drawn from the distribution $\mathcal{D}$, and in particular the entire simulated sample $\{(\boldsymbol{z}'_i, y'_i) : i \in [n^s]\}$, are realizable by $R$. That is, $\forall i \in [n^s], \quad \boldsymbol{z}'_i \in L(R) \iff y'_i = 1$.

Fix the seed $\boldsymbol{x} \in \{0,1\}^n$ for the pseudorandom generator. Without loss of generality, assume $n$ is a power of two, thus, the block length $\log(n)$ is an integer and every $\log(n)$-bit block encodes some index in $[n]$. Let $\mathrm{bin}(i)$ be the binary representation of $i \in [n]$. For $b \in \{0,1\}$ define

$$I_b := \bigvee_{i \in [n]:\ x_i = b} \mathrm{bin}(i),$$

that is, $I_b$ matches exactly those $\log(n)$-bit blocks corresponding to indices $i$ with $x_i = b$. For each $\boldsymbol{u} = (u_1, \ldots, u_k) \in \{0,1\}^k$ with $P(\boldsymbol{u}) = 1$ set

$$R_{\boldsymbol{u}} := I_{u_1} I_{u_2} \cdots I_{u_k} (0 \vee 1)^\star.$$

Let

$$R_{\boldsymbol{x}} := \bigvee_{\boldsymbol{u}:\ P(\boldsymbol{u})=1} R_{\boldsymbol{u}}.$$

This is a regex that accepts exactly those encodings of hyperedges $(i_1, \ldots, i_k)$ where $P(x_{i_1}, \ldots, x_{i_k}) = 1$.

Since examples drawn from $\mathcal{D}$ are not necessarily extended compressed encodings but still have label 1, we need an additional regular expression to capture them. This occurs when an index appears more than once. We define this duplicate-catcher regular expression as

$$R_{\mathrm{dup}} := \bigvee_{1 \le a < b \le k} \bigvee_{i \in [n]} (0 \vee 1)^{(a-1)\log(n)} \mathrm{bin}(i) (0 \vee 1)^{(b-a-1)\log(n)} \mathrm{bin}(i) (0 \vee 1)^\star.$$

Note that we use $(0 \vee 1)^\star$ instead of a fixed-length suffix in both $R_{\mathrm{dup}}$ and $R_{\boldsymbol{u}}$, since our distribution $\mathcal{D}$ is over strings of length $n^\alpha$, and the behavior on other lengths is irrelevant. Finally, define the target regex

$$R := R_{\boldsymbol{x}} \vee R_{\mathrm{dup}}.$$

If $\boldsymbol{z}'_i$ encodes a valid hyperedge $E = (i_1, \ldots, i_k)$ then each block $I_{u_j}$ enforces $x_{i_j} = u_j$. Hence $R_{\boldsymbol{x}}(\boldsymbol{z}'_i) = P(\boldsymbol{x}|_E)$, and $R(\boldsymbol{z}'_i) = P(\boldsymbol{x}|_E)$. If the encoding is invalid (some index repeats), then $R_{\mathrm{dup}}$ accepts and $R(\boldsymbol{z}'_i) = 1$, matching the simulated labeling rule. Note that the inputs to $R$ are in $\{0,1\}^{n^\alpha}$, but it uses only the first $k \log(n)$ coordinates of the input. The construction clearly runs in polynomial time.

**The size of $R$.** Each $I_b$ is a union of at most $O(n)$ strings of length $\log(n)$, so $|I_b| = O(n \log(n))$. Thus, for each pattern $\boldsymbol{u}$ we obtain $|R_{\boldsymbol{u}}| = O(kn \log(n))$. Taking the union over at most $2^k$ such patterns gives $|R_{\boldsymbol{x}}| = O(2^k kn \log(n))$. For the duplicate–catcher, each of the $\binom{k}{2} n = O(k^2 n)$ branches. Each branch has size $O(k \log(n))$: two strings of length $\log(n)$ and two fixed gaps whose total length is at most $(k-2)\log(n)$. Thus, $|R_{\mathrm{dup}}| \le O(k^3 n \log(n))$. Therefore, the total size is

$$|R| = |R_{\boldsymbol{x}}| + |R_{\mathrm{dup}}| = O(2^k kn \log(n)) + O(k^3 n \log(n)).$$

Since $k$ is independent of $n$, for sufficiently large $n$ we get $|R| \le O(n \log^2(n))$.

**Analysis (pseudorandom case).** If $\boldsymbol{E}$ is pseudorandom, then for every oracle answer we have $y_i' = R(\boldsymbol{z}_i')$. Thus with probability at least $\frac{3}{4}$, algorithm $L$ outputs $h$ with $\mathbb{E}_{\tilde{\boldsymbol{z}}\sim\mathcal{D}}[\mathbb{I}\{h(\tilde{\boldsymbol{z}}) \neq R(\tilde{\boldsymbol{z}})\}] \leq \frac{1}{2} - \gamma$. Thus, $\mathbb{E}_{S_v}[\widehat{\mathtt{Err}}_{S_v}(h)] \leq \frac{1}{2} - \gamma$. By Hoeffding's inequality, $\mathbb{P}_{S_v}\left[\widehat{\mathtt{Err}}_{S_v}(h) - \mathbb{E}_{S_v}[\widehat{\mathtt{Err}}_{S_v}(h)] \geq \frac{\gamma}{4}\right] \leq \frac{1}{20}$. Therefore, with probability at least $1 - \left(\frac{1}{4} + \frac{1}{20}\right) = \frac{7}{10} > \frac{2}{3}$, we get $\widehat{\mathtt{Err}}_{S_v}(h) \leq \frac{1}{2} - \frac{\gamma}{2}$, so $A$ outputs 1.

**Analysis (random case).** If $\boldsymbol{E}$ is random, then whenever $\boldsymbol{z}_i'$ encodes a hyperedge, the label $y_i'$ is independent of $\boldsymbol{z}_i'$ and uniform. Thus for every $h$ and $(\boldsymbol{z}_i', y_i') \in S_v$,

$$\begin{aligned}\mathbb{P}[h(\boldsymbol{z}_i') \neq y_i'] &\geq \mathbb{P}[h(\boldsymbol{z}_i') \neq y_i' \mid \boldsymbol{z}_i' \text{ encodes a hyperedge}] \cdot \mathbb{P}[\boldsymbol{z}_i' \text{ encodes a hyperedge}] \\ &\geq \tfrac{1}{2} \cdot (1 - \tfrac{\gamma}{2}) \\ &= \tfrac{1}{2} - \tfrac{\gamma}{4},\end{aligned}$$

and $\mathbb{E}_{S_v}[\widehat{\mathtt{Err}}_{S_v}(h)] \geq \frac{1}{2} - \frac{\gamma}{4}$. Applying Hoeffding's inequality again, with probability at least $\frac{19}{20}$ we have $\widehat{\mathtt{Err}}_{S_v}(h) \geq \frac{1}{2} - \frac{\gamma}{2}$, so $A$ outputs 0 with probability at least $\frac{2}{3}$.

**Conclusion.** Thus $A$ distinguishes pseudorandom from random with advantage $> 1/3$, contradicting Assumption 4.3. Hence, for sufficiently large $n \in \mathbb{N}$, efficient weak learning of regular expressions of size $n\log^2(n)$ is impossible under the uniform distribution over $\{0,1\}^{n^\alpha}$. We now follow a standard scaling argument (see e.g. Daniely et al. (2014) to obtain a tighter result.

Fix any $\epsilon > 0$. Choose $\alpha = 1 + \frac{2}{\epsilon}$, and let $\tilde{n} = n^\alpha = n^{1+\frac{2}{\epsilon}}$. Thus, no polynomial-time learner weakly learns regular expressions of most $\tilde{n}^\epsilon = n^{2+\epsilon}$ (which already subsumes $n\log^2(n)$), under the uniform distribution over $\{0,1\}^{\tilde{n}}$.

**Extensions to RE without Kleene star, and RE with the counting operator:**

- **RE with union and concatenation (without Kleene star).** We can construct a similar star-free RE as follows. $I_b$ remains the same. $R_{\boldsymbol{u}}$ is modified to

$$R_{\boldsymbol{u}} := I_{u_1} I_{u_2} \cdots I_{u_k},$$

and

$$R_{\boldsymbol{x}} := \left( \bigvee_{\boldsymbol{u}:\, P(\boldsymbol{u})=1} R_{\boldsymbol{u}} \right) (0 \vee 1)^{(n^\alpha - k\log(n))}.$$

We modify $R_{\text{dup}}$ as follows. For $1 \leq a < b \leq k$ and $i \in [n]$ define

$$R_{\text{dup}(a,b,i)} := (0 \vee 1)^{(a-1)\log(n)} \text{bin}(i) (0 \vee 1)^{(b-a-1)\log(n)} \text{bin}(i) (0 \vee 1)^{(k-b)\log(n)},$$

which means that the $a$-th and $b$-th blocks of size $\log(n)$ are the same index $i$. Then

$$R_{\text{dup}} := \left( \bigvee_{1 \leq a < b \leq k} \bigvee_{i \in [n]} R_{\text{dup}(a,b,i)} \right) (0 \vee 1)^{n^\alpha - k\log(n)}.$$

The final star-free RE is $R := R_{\boldsymbol{x}} \vee R_{\text{dup}}$.

However, we get a larger RE. Each block matcher $I_b$ is a union of at most $O(n)$ strings of length $\log(n)$, so $|I_b| = O(n\log(n))$. Thus, for each pattern $\boldsymbol{u}$ we obtain $|R_{\boldsymbol{u}}| = O(kn\log(n))$. Taking the union over at most $2^k$ such patterns gives $|R_{\boldsymbol{x}}| = O(2^k kn\log(n)) + O(n^\alpha)$. Note that the suffix of size $O(n^\alpha)$ is added to $R_{\boldsymbol{x}}$ only once. For the duplicate–catcher, each of the $\binom{k}{2}n = O(k^2 n)$ branches has size $k\log(n)$, and in addition, a suffix of size $O(n^\alpha)$. Hence, $|R_{\text{dup}}| = O(k^3 n \cdot \log(n)) + O(n^\alpha)$. Therefore, the total size is

$$|R| = |R_{\boldsymbol{x}}| + |R_{\text{dup}}| = O(2^k kn\log(n)) + O(n^\alpha) + O(k^3 n \cdot \log(n)) + O(n^\alpha).$$

Choosing $\alpha = 2$, we get $|R| = O(n^2)$.

Then, efficient weak learning of star-free regular expressions of size $O(n^2)$ is impossible under the uniform distribution over $\{0,1\}^{n^2}$. By taking $\tilde{n} = n^2$, we conclude that there is no polynomial-time learner that weakly learns, under the uniform distribution over $\{0,1\}^{\tilde{n}}$, star-free regular expressions of size at most $O(\tilde{n})$. Note that without the Kleene star or the counting operator, we cannot expect to get a better result than a regular expression of linear size.

- **RE with union, concatenation, and counting (without Kleene star).** We can reduce the size of the star-free RE by using the counting operator. With counting, the suffix $(0 \vee 1)^{n^\alpha - k\log(n)}$ is described by $O(\log(n))$ symbols, rather than $O(n^\alpha)$ repetitions and we get $|R_{\boldsymbol{x}}| = O(2^k kn\log(n))$. Similarly, for the duplicate–catcher, the suffix is also $O(\log(n))$ and $|R_{\text{dup}}| = O(k^3 n\log(n))$. Therefore the total size is $|R| = O(2^k kn\log(n)) + O(k^3 n\log(n))$, and for sufficiently large $n$ simplifies to $|R| = O(n\log^2(n))$.

  The size of the regex is as in the construction of the plain RE (with Kleene star), so we conclude that there is no polynomial-time learner that weakly learns regular expressions of most $\tilde{n}^\epsilon$ under the uniform distribution over $\{0,1\}^{\tilde{n}}$.

■