Learning Expressions over Monoids (Extended Abstract)

Ricard Gavaldà[1⋆] and Denis Thérien[2⋆⋆]

[1] Department of Software (LSI), Universitat Politècnica de Catalunya
gavalda@lsi.upc.es
[2] School of Computer Science, McGill University
denis@cs.mcgill.ca

Abstract. We study the problem of learning an unknown function represented as an expression over a known finite monoid. As in other areas of computational complexity where programs over algebras have been used, the goal is to relate the computational complexity of the learning problem with the algebraic complexity of the finite monoid. Indeed, our results indicate a close connection between both kinds of complexity. We focus on monoids which are either groups or aperiodic, and on the learning model of exact learning from queries. For a group G, we prove that expressions over G are easily learnable if G is nilpotent and impossible to learn efficiently (under cryptographic assumptions) if G is nonsolvable. We present some partial results for solvable groups, and point out a connection between their efficient learnability and the existence of lower bounds on their computational power in the program model. For aperiodic monoids, our results seem to indicate that the monoid class known as DA captures exactly learnability of expressions by polynomially many Evaluation queries.

1 Introduction

Formal models of the process of learning have been proposed since the 60s to give mathematical foundation to machine learning tasks, mostly to concept learning. The first models, such as Gold's identification in the limit, were of recursion-theoretic flavor and did not emphasize efficient use of computational resources. In the mid 80s, several models that take time and memory into consideration were proposed, thus allowing the use of concepts and tools from computational complexity theory in the study of learnability.

The resulting area, known as Computational Learning Theory, has produced an important number of results stating that certain classes of functions are or are

⋆ Partially supported by the EC ESPRIT Working group NeuroCOLT2, the EC ESPRIT AlcomFT project, and by FRESCO PB98-0937-C04-04. Part of this research was done while this author visited McGill University, supported by NSERC and FCAR.

⋆⋆ Partially supported by NSERC and FCAR grants and by the von Humboldt Foundation.

not efficiently learnable in rigorously defined learning models. But, in general, the properties that determine whether a class is easy or hard to learn are not yet well identified. We propose an algebraic approach that, in the long run, might help in clarifying such properties.

Programs over finite monoids and other algebraic structures are models of computation that have been successfully used to expose the deep reasons why some computational problems can or cannot be solved within certain resources. In this paper we initiate the study of formal models of function learning from an algebraic point of view, i.e., we would like to determine the complexity of learning a class of functions from the classes of algebras that are powerful enough to compute the class of functions (in the program model or related ones). We concentrate on the case where the algebra M consists of an associative operation with an identity, i.e., when M is a monoid.

So far, programs over monoids have been studied mostly as devices to compute boolean functions, and many circuit complexity classes have been shown to admit characterizations as those problems solved by programs over particular algebraic structures [4, 6, 5]. To avoid replicating this study, we look instead at programs over M as computing functions from M^n into M. *Expressions* over M are a particular type of programs that appear very naturally in this context.

We study the problem of learning an unknown *target* function from M^n to M, for a fixed and known finite monoid M. It is assumed only that the function is computed by some expression on n variables over M. We work mostly in Angluin's query-based model of exact learning [1, 2], where algorithms can ask Evaluation queries, or Equivalence queries, or both.

Note that this problem is not obviously comparable to the problem of learning the class of boolean circuits corresponding to programs over monoid M. This is because, on the one hand, the problem might be harder because the function class is richer. On the other hand, the answers to the queries provide finer information than in the boolean setting, namely, elements of the monoid, and this could help in learning.

We present several results on the complexity of learning expressions over specific classes of monoids. We concentrate on monoids that are either *groups* or *aperiodic*, as these two classes are known to be the building blocks of all monoids via the so-called *wreath product* operation.

Along the paper, we often say "a monoid M is learnable" for the sake of brevity, meaning "expressions over monoid M are learnable", and similarly for a class of monoids.

For the case of groups, we prove:

- Expressions over Abelian groups are polynomial-time learnable both from a linear number of Evaluation queries and from a linear number of Equivalence queries.
- Nilpotent groups (a generalization of Abelian groups) are learnable in polynomial time from Evaluation queries.
- Solvable groups formed as extensions of a group Z_p by a group Z_q (for any p and q) are polynomial-time learnable from Evaluation and Equivalence

queries in the form of Multiplicity Automata, and learnable in probabilistic polynomial time from Evaluation queries alone.
- A slightly larger subclass of solvable groups can be identified by a probabilistic strategy using polynomially many Evaluation queries, and by a deterministic strategy using quasipolynomially many Evaluation queries (though no claims are made regarding computation time). We show that, in fact, these or similar results will hold for any group for which we can prove a certain type of lower bound on its computation power; in other words, such lower bounds on computation power provide upper bounds on learning complexity.
- Expressions over nonsolvable groups are not learnable unless NC^1 circuits also are, even in the very strong learning model of PAC-prediction with Membership queries [3]. Recall that under plausible cryptographic assumptions, NC^1 circuits are not learnable in this or any other standard model of learning [3].

In the aperiodic case, our results involve the class of aperiodic monoids known as DA [15]. For some algorithmic problems on monoids, it is known that feasibility depends essentially on membership to DA. For example, the membership problem is known to be PSPACE-complete for any aperiodic monoid outside of DA [7]. Also, the word problem for an aperiodic monoid can be resolved in sublinear communication complexity (in the 2-player setting) iff the monoid belongs to DA [14]. Our results are:

- Expressions over DA monoids are identifiable from polynomially many Evaluation queries (though possibly not in polynomial time).
- For a subclass of DA, idempotent R-trivial monoids, we can in fact give a polynomial-time learning algorithm using Evaluation queries.
- It is known that there are exactly two minimal aperiodic monoids not belonging to DA. We show that expressions over any of these two minimal monoids are not learnable with subexponentially many Evaluation queries, even with arbitrary computation time. We conjecture the same is true for any monoid outside of DA, because it is also known that every monoid outside of DA is divided by at least one of these two minimal monoids.

Certainly the picture is still partial, with many upper and lower bounds missing. But our results seem to indicate a very close connection between the complexity of the monoid (in the algebraic sense) and its learning complexity.

Finally, let us comment on the relevance of these results for the mainstream of computational learning theory. A good deal of the effort in this theory has been on learning classes of boolean functions, and especially those inside NC^1 since it seems hopeless to try to learn any larger one. As mentioned, many of the central complexity classes definable by small-depth circuits can be also defined by programs over central classes of monoids. For example, polynomial-length programs over aperiodic monoids compute exactly the functions in AC^0, programs over solvable groups compute the functions in CC^0 (polynomial-size circuits made of mod_q gates for a fixed q), and nonsolvable groups (or monoids) compute all functions in NC^1 [4, 6]. Formally, we do not know how to translate neither positive nor negative results in our model to the boolean case. On the other hand, in the learning context, various small fragments of AC^0 and CC^0 are known to be learnable, and NC^1 is known not be learnable in a strong sense, in fact a situation quite in analogy with the results in this paper. We believe they are interesting as they provide a new perspective on learning problems near the borderline of current knowledge.

Due to lack of space, this extended abstract does not contain the proofs of our results. The full version of the paper can be found on the respective home pages of the authors, http://www.lsi.upc.es/gavalda and http://www.cs.mcgill.ca/denis.

2 Preliminaries

2.1 Expressions and Programs over Monoids

A *monoid* is a pair $(M, \cdot)$ where M is a set and $\cdot$, the product over M, is a binary associative operation on M with an identity element. A *group* is a monoid where each element has a (two-sided) inverse with respect to $\cdot$. A monoid is *aperiodic* if it has no submonoid which is a non-trivial group. All monoids considered in this paper will be finite. We look at two computation models based on products over a monoid: expressions and programs.

An *expression* over monoid M and variables $x_1, \dots, x_n$ is a string over the alphabet $M \cup \{x_1, \dots, x_n\}$. Such an expression defines quite naturally a function from M^n to M: to evaluate the (function represented by the) expression over a vector or assignment $(w_1, w_2, \dots, w_n)$ in M^n, replace with w_i each occurrence of each variable x_i in the expression, then multiply out the resulting string of monoid elements to obtain a single monoid element. For example, assume that a, b, c, d are four elements in M. Then the value of expression $ax_2x_1x_2bx_3cx_1x_3d$ on the assignment (c, b, a) is the element $a \cdot b \cdot c \cdot b \cdot b \cdot a \cdot c \cdot c \cdot a \cdot d$, where $\cdot$ is the product in M.

Expressions are a particular case of programs. A *program* over M with domain D is a list of instructions of the form (i, f), where $i \in \{1, \dots, n\}$ and f is a function $D \mapsto M$. Instruction (i, f) is interpreted as follows: read the value of variable x_i and append $f(x_i)$ to the string of monoid elements to be multiplied. Hence, expressions are programs whose instructions use only constant functions and the identity function.

In the literature, programs have been used mostly to compute boolean functions [4, 6]. In these *boolean programs*, domain D is {true,false} and M is partitioned into True and False sets to interpret a boolean result. Expressions could also be used to compute boolean functions, say by encoding true and false inputs by distinct elements of the monoid. We will be here mainly interested in programs and expressions computing functions from M^n into M, rather than boolean functions.

2.2 Learning Expressions over Monoids: Problem Statement

In this section we define the learning problem we study. We use mostly Angluin's query-based model of learning [1, 2], although Valiant's PAC model [17] or related ones are mentioned occasionally. All definitions are standard for function learning in these models, although we give them only in the terms of our specific problem, learning expressions over monoids. For background in computational learning theory, the reader is referred to the books, surveys, and bibliography in the recently created server [13].

The task of a learning algorithm (or *learner*) is to find out a *target function* M^n to M fixed by a *teacher* in an adversary way. The function is assumed to be representable as an expression over M and variables $x_1, \dots, x_n$, though not all variables are necessarily used. The learning algorithm is initially given n and some upper bound m on the length of a shortest expression for the target.[1] Monoid M is fixed and known both to the teacher and the learning algorithm.

The learning algorithm must produce an expression (or some other representation of the target function, as discussed later) equivalent to the target one on all of M^n.

To achieve learning, teacher and learner exchange information on the target function following some protocol; some specific protocols will be defined later.

Resources used by the algorithm are measured as a function of n and m. The class of expressions over M is learnable in time $t(n,m)$ in a given learning protocol if there is an algorithm that learns every expression over M in time $t(n,m)$. In particular, we study mostly whether expressions over a class of monoids is *polynomial-time learnable*, meaning whether for each monoid M in the class there is a polynomial $p(n,m)$ such that expressions over M are learnable in time $p(n,m)$.

Similarly, we say that expressions over a monoid M are *identifiable* with interaction $s(n,m)$ if there is an algorithm that learns every expression over M using an amount of interaction $s(n,m)$ with the teacher (and arbitrary computation time). The meaning of "amount of interaction" may be different in each learning protocol, but in general it has to be bounded by the number of bits of information exchanged by the teacher and the learning algorithm. Identifiability thus represents the information-theoretic cost of solving a learning task, ignoring the computational complexity of the problems that the learning algorithm has to solve internally at each stage of the process.

Let us stress that we assume that monoid M is fixed and known to the learning algorithm, and thus we regard $|M|$ as a constant. We often present algorithmic schemes to learn whole classes of monoids whose running time is exponential (or more) in the size of the monoid. We still call these algorithms "polynomial-time" as long as their dependence on n and m is polynomial. A stronger notion of "polynomial-time learnability" of a monoid class would ask for an algorithmic scheme depending only polynomially on $|M|$. In an even stricter sense, we could ask for an algorithm that receives the multiplication table of M as part of the input, with the promise that M belongs to the monoid class, and is polynomial in $|M|$, n, and m; this would be truly "uniformly" learning

[1] Symbols n and m will always have this meaning in the paper, i.e., number of variables and an upper bound on the length of shortest expression for the target.

of class, in the sense that the algorithm does not rely on hardwired information for each monoid.

As for interaction between teacher and learner, we consider two standard query types in Angluin's model: Evaluation and Equivalence queries. Let the target f be a function from set A to set B. In an Evaluation query, the learning algorithm produces an element $a \in A$ and the teacher must return $f(a)$.[2] In an Equivalence query, the learning algorithm produces a hypothesis h, representing a function $A \mapsto B$ in some way; the teacher must return Yes if $h \equiv f$ on A, or else a counterexample: an element $a \in A$ such that $f(a) \neq h(a)$, together with the value of $f(a)$. If hypotheses issued by the algorithm always belong to the same syntactic class of functions that is being learned, the algorithm is called *proper*. Otherwise, hypotheses may belong to a different and possibly richer class, and the algorithm is called *nonproper*. An important requirement on any such hypothesis class is that it must be polynomial-time evaluatable, i.e., that a given hypothesis can be evaluated on a given input in polynomial time.

Before investigating the learnability of specific classes of monoids, let us consider a quite general question. There are several constructions for building monoids from other monoids. The most natural ones are direct product, submonoid, and homomorphic image. It is a natural question whether learnability is preserved by these operations. We only have very partial answers so far.

Proposition 1.

1. *For the three models of query learning (Evaluation queries only, Equivalence queries only, or both) the following is true. If expressions over monoids S and T are polynomial-time learnable, then expressions over $S \times T$ are learnable (possibly nonproperly).*
2. *If expressions over T are polynomial-time learnable from Equivalence queries and S is a submonoid of T, then expressions over S are polynomial-time learnable from Equivalence queries.*

We do not know whether learnability under Evaluation queries is preserved by taking submonoids. The difficulty is that the algorithm for the larger monoid may ask queries on $T^n \setminus S^n$; the teacher, knowing only a target function $S^n \mapsto S$, is not able to answer these. In fact, the query is ill-posed as the answer may be different for different T-expressions defining the same target function over S. Under homomorphic image and either type of query, the problem lies in inverting the homomorphism on answers to queries in a way that is guaranteed to be consistent with some expression over the larger monoid.

3 Abelian Groups

Abelian or commutative groups are the simplest from an algebraic point of view. Quite naturally, they are easiest from the learning point of view, in the sense

[2] Evaluation queries generalize Membership queries [1, 2] for functions with non-binary range.

that they are learnable with a linear number of either Evaluation or Equivalence queries.

Theorem 1.

a) Expressions over an Abelian group G are learnable with $n+1$ Evaluation queries.
b) Expressions over an Abelian group G are learnable with $O(n)$ Equivalence queries.

4 Nilpotent groups

Let a and b be elements of a group G and let $[a,b] = aba^{-1}b^{-1}$ denote the commutator of a and b. These are the commutators of weight 2. Commutators of weight 3 are $[a,[b,c]]$ and $[[a,b],c]$ and commutators of weight k are defined inductively in the obvious way. We say that G is nilpotent of class-k iff all commutators of weight $k+1$ are the identity, and observe that any commutator of any weight involving the identity it itself the identity.

It is clear that nilpotent groups of class 1 are exactly the Abelian groups. And indeed several properties of nilpotent groups are natural generalizations of those for Abelian groups. For example, it can be shown that n-variable functions that are realizable by programs over nilpotent groups of class k can always be represented (in the sense of [5]) by polynomials of degree k (with coefficients in an appropriate ring). Expressions over nilpotent groups are learnable from Evaluation queries alone. As in the Abelian case, the learning algorithm is based on the fact that programs can be rewritten to a normal form, although the transformation is more involved in this case.

Theorem 2. *Expressions over a nilpotent class-k group G are learnable with $|G|^k n^k$ Evaluation queries and $n^k |G|^{O(|G|^k)}$ time.*

For Equivalence queries, an approach like that in Theorem 1, part (b), seems difficult since it would involve solving polynomial equations over cyclic groups.

5 Solvable Groups

In this section we will present some partial results on learnability in solvable groups. For any two subsets A and B of a group G, denote by $[A,B]$ the subgroup generated by all commutators $[a,b]$, with $a \in A, b \in B$. We can then form the descending chain of subgroups $G_0, G_1, \ldots$ by setting $G_0 = G$ and $G_i = [G_{i-1}, G_{i-1}]$. The group G is solvable iff this chain goes to the trivial subgroup.

In order to present our results concerning solvable groups, we have to recall the following notion. Let G be a group, and suppose H is a normal subgroup of G. Then G is said to be an extension of H by the quotient group G/H and it admits the following representation.

We view elements of G as pairs in $H \times G/H$. For any pairs $(h_1, g_1), (h_2, g_2) \in G$, product in G can be expressed as:

$$(h_1, g_1) \cdot_G (h_2, g_2) = (h_1 \cdot_H f_{g_1,g_2}(h_2), g_1 \cdot_{G/H} g_2).$$

Functions $f_{g_1,g_2} : H \mapsto H$ for each $g_1, g_2 \in G/H$ are called the "twist functions" for G and describe the interaction between the two components of the group.

Our results in this section concern solvable groups which are extensions of Z_p by Z_q, where Z_p and Z_q are any cyclic groups. Note that, for example, the group S_3 of permutations on three points is an extension of Z_3 by Z_2.

From now on, we view elements of a group G as above as pairs in $Z_p \times Z_q$. For an element $g \in G$, we use notations $Z_p(g)$ and $Z_q(g)$ to denote the first and second elements of the pair associated to g, that is, we identify g with the pair $(Z_p(g), Z_q(g))$.

The learning algorithm for these groups uses Multiplicity Automata as hypothesis class. Multiplicity Automata over rings (MA for short) are an important generalization of classical automata. They were first used in the context of learning by Bergadano and Varricchio [10], who gave a polynomial-time algorithm for learning MA over fields by Equivalence and Evaluation (there called Multiplicity) queries. Later, Bshouty, Tamon, and Wilson [12] extended the algorithm to work over a large class of rings instead of fields, including all finite integer rings.

The algorithm for MA has been used to learn several other classes of functions [9, 8]. In particular, [9] uses the MA learning algorithm to learn some classes of boolean circuits with modular gates and boolean permutation branching programs of width at most 4. These results are probably related to the connection between some solvable groups and MA that we find here.

We give here a working definition of Multiplicity Automata; for more systematic presentations see [10, 12]. An MA over an alphabet Σ and a ring K is a nondeterministic finite automata where each transition triple (q, a, q') $(a \in \Sigma)$ is additionally labeled by an element of K. To each path in the automata we associate the value in K given by the product of all the labels along the path. The MA computes a function $M : \Sigma^\star \mapsto K$ in the following way: for each input $w \in \Sigma^\star$, $M(w)$ is the sum of the values of all nondeterministic paths defined by input w on the MA. A particular case of MA that we use here is when $\Sigma = K$ and the MA is evaluated on inputs of a fixed length n, so that $M : K^n \mapsto K$.

Multiplicity Automata are able to simulate expressions over the groups above. The proof is based on a somewhat careful study of the structure of their twist functions.

Theorem 3. *Let G be a group which is an extension of Z_p by Z_q. Then there is a function $f : Z_q \mapsto Z_p$ such that for every expression $E(x_1, \ldots, x_n)$ over G there is a multiplicity automata M over Z_p of size $O(n|E|)$ such that for all $a_1, \ldots, a_n \in G$,*

$$Z_p(E(a_1, \ldots, a_n)) = M(Z_p(a_1), \ldots, Z_p(a_n), f(Z_q(a_1)), \ldots, f(Z_q(a_n))).$$

Furthermore, for all vectors $u_1, \ldots, u_n \in Z_p$ and $v_1, \ldots, v_n \in Z_p$ such that some v_i is not in the range of f,

$$M(u_1, \ldots, u_n, v_1, \ldots, v_n) = 0.$$

Then, using the learning algorithm for MA over rings [12], we can prove:

Theorem 4. *For every group G as above, expressions over G are learnable in polynomial time using Evaluation and Equivalence queries. The Equivalence queries and the output of the algorithm are pairs formed by a Multiplicity Automata over the ring Z_p and an expression over Z_q.*

When p is prime, the groups we have been working with in this section can in particular be viewed as special cases of wreath products of abelian groups by p-groups. This larger family has been studied in [5] where it was shown that any group of that form could not possibly compute the AND function via a program of subexponential length. Moreover, these groups were also shown to have the following property.

Say that a solvable group G is non-narrowing if there is a polynomial $p(m)$ such that for all program P and all $a \in G$, if there is an assignment $w \in G^n$ such that $P(w) = a$ then there are at least $|G|^n / p(length(P))$ such assignments. This property gives an identification strategy by polynomially many Evaluation queries (not necessarily a polynomial-time algorithm as we don't know how to efficiently obtain a hypothesis from the answers).

The remaining results in this section were obtained through discussions with Cris Moore.

Theorem 5. *If G is non-narrowing, then programs (hence, expressions) over G can be identified probabilistically from Evaluation queries in polynomial time.*

With the same argument, it is easy to show that an Equivalence query to the groups above can be simulated with high probability by polynomially many random Evaluation queries. We can combine this observation with the Equivalence and Evaluation query algorithm in Theorem 4.

Corollary 1. *Any solvable group which is an extension of Z_p (p prime) by Z_q is learnable from Evaluation queries in probabilistic polynomial time.*

We finally observe that an exponential lower bound on the length of programs over G that compute the AND function translates into a quasipolynomial upper bound on the number of Evaluation queries needed to identify a program over that group.

Theorem 6. *If programs over G cannot compute the AND function in subexponential length, then programs over G can be identified from $n^{O(\log m)}$ Evaluation queries.*

Note that it is conjectured in [5] that the exponential lower bound on the AND function holds for all solvable groups.

6 Hardness of Nonsolvable Groups

In this section we now show that expressions over nonsolvable groups are not polynomial-time learnable unless NC^1 circuits are polynomial-time learnable too. Hence, under the cryptographic assumptions in [3], they are not learnable at all.

Theorem 7. *Expressions over nonsolvable groups are not learnable from Evaluation and Equivalence queries with any polynomial-time evaluatable class, unless NC^1 circuits also are*

The proof works by reducing learnability of nonsolvable groups to that of boolean programs over simple nonabelian groups, which are polynomially equivalent to NC^1 circuits [4].

Conceptually, there are three parts in the reduction: 1) learning expressions over some nonsolvable group implies learning expressions over some simple nonabelian group; this does not follow trivially from the fact that a nonsolvable group contains a simple nonabelian one, because the learning algorithm for nonsolvable might conceivably use queries outside the simple nonabelian one to learn it; 2) over a simple nonabelian group, programs and expressions compute the same functions; 3) over a simple nonabelian group, programs simulate boolean programs in a prediction-preserving sense.

7 Aperiodic Monoids

Theorem 8. *Expressions over a monoid in DA are learnable from a polynomial number of Evaluation queries and unbounded computation time.*

Although the answers to these many Evaluation queries identify uniquely the target, they give no obvious way to predict the value of the target on a different input. For a small subclass of DA we know how to reconstruct efficiently an expression for the target from these answers.

Theorem 9. *Expressions over idempotent R-trivial monoids are learnable in polynomial time from Evaluation queries. The same is true for aperiodic commutative monoids.*

It is known that there are exactly two minimal monoids outside of DA, named U and BA_2. Monoid U is the syntactic monoid of the language $(aa^\star b)^\star$, has 6 elements, and is known to be universal because programs over it can simulate DNF formulas [16]. Monoid BA_2 is the syntactic monoid of $(ab)^\star$, has 6 elements also, and is provably not universal.

We reduce the problem of learning expressions over U from Evaluation queries reduces to the problems of learning monotone DNF formulas from Membership queries, which requires exponentially many queries [11]. Similarly, learning expressions over BA_2 reduces to learning singleton sets by Membership queries.

Theorem 10. *Expressions over monoid U are not learnable from subexponentially many Evaluation queries, even using unbounded computation time. The same holds for monoid BA_2.*

8 Acknowledgements

We are indebted to Cris Moore for ideas and discussions that lead to Theorems 5, 1, and 6. We thank Pascal Tesson and José L. Balcázar for helpful discussions.

References

1. D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75:87–106, 1987.
2. D. Angluin. Queries and concept learning. *Machine Learning*, 2:319–342, 1988.
3. D. Angluin and M. Kharitonov. When won't membership queries help? *Journal of Computer and System Sciences*, 50:336–355, 1995.
4. D. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *Journal of Computer and System Sciences*, 38:150–164, 1989.
5. D. M. Barrington, H. Straubing, and D. Thérien. Non-uniform automata over groups. *Information and Computation*, 89:109–132, 1990.
6. D. M. Barrington and D. Thérien. Finite monoids and the fine structure of NC^1. *Journal of the ACM*, 35:941–952, 1988.
7. M. Beaudry, P. McKenzie, and D. Thérien. The membership problem in aperiodic transformation monoids. *Journal of the ACM*, 39(3):599–616, 1992.
8. A. Beimel, F. Bergadano, N. Bshouty, E. Kushilevitz, and S. Varricchio. Learning functions represented as multiplicity automata. *Journal of the ACM*, 47:506–530, 2000.
9. F. Bergadano, N. Bshouty, C. Tamon, and S. Varricchio. On learning branching programs and small depth circuits. *Proc. 3rd European Conference on Computational Learning Theory (EuroCOLT'97), Springer-Verlag LNCS*, 1208:150–161, 1997.
10. F. Bergadano and S. Varricchio. Learning behaviors of automata from multiplicity and equivalence queries. *SIAM Journal on Computing*, 25:1268–1280, 1996.
11. N. Bshouty, R. Cleve, R. Gavaldà, S. Kannan, and C. Tamon. Oracles and queries that are sufficient for exact learning. *Journal of Computer and System Sciences*, 52:421–433, 1996.
12. N. Bshouty, C. Tamon, and D. Wilson. Learning matrix functions over rings. *Algorithmica*, 22:91–111, 1998.
13. The learning theory server. `www.learningtheory.org`.
14. J.-F. Raymond, P. Tesson, and D. Thérien. An algebraic approach to communication complexity. *Proc. ICALP'98, Springer-Verlag LNCS*, 1443:29–40, 1998.
15. M. Schützenberger. Sur le produit de concaténation non ambigu. *Semigroup Forum*, 13:47–75, 1976.
16. D. Thérien. Programs over aperiodic monoids. *Theoretical Computer Science*, 64(3):271–280, 29 1989.
17. L. Valiant. A theory of the learnable. *Communications of the ACM*, 27:1134–1142, 1984.