On the Power of Equivalence Queries

Ricard Gavaldà *

Department of Software (LSI)
Universitat Politècnica de Catalunya
Pau Gargallo, 5
08028 Barcelona, Spain

June 20th, 1994

Abstract

In 1990, Angluin showed that no class exhibiting a combinatorial property called "approximate fingerprints" can be identified exactly using polynomially many Equivalence queries (of polynomial size). Here we show that this is a necessary condition: every class without approximate fingerprints has an identification strategy that makes a polynomial number of Equivalence queries. Furthermore, if the class is "honest" in a technical sense, the computational power required by the strategy is within the polynomial-time hierarchy, so proving nonlearnability is at least as hard as showing P $\neq$ NP.

1 Introduction

Learning via queries is a well-studied model in computational learning. The types of queries that have been used most often in the design of learning algorithms are, by far, Membership and Equivalence queries. In this paper we focus on the second type.

*This research was partially supported by the ESPRIT Basic Research Actions Program of the EC under contract No. 7141 (project ALCOM II), and by DAAD and MEC through Acciones Integradas 1992, 131-B, 313-AI-e-es/zk. E-mail: gavalda@lsi.upc.es

There are a few classes of concepts known to be learnable from Equivalence queries alone: CNF and DNF formulas with a bounded number of literals per clause [2], discrete geometric concepts [10], and some types of simple grammars [14]. (By "learnable" we always mean "learnable in polynomial time"; see Section 2 for definitions.)

Equivalence queries, however, are provably insufficient for learning many other classes. Angluin [3] showed that they do not suffice for learning deterministic or nondeterministic finite automata, context-free grammars, or general CNF or DNF formulas. She proved these facts by showing that these classes exhibit a certain combinatorial property, "having approximate fingerprints", together with the following theorem.

Theorem 1 [3] *If a representation class $\mathcal{C}$ has approximate fingerprints, then it is not learnable from Equivalence queries.*

It is important to realize that the proof of Theorem 1 relies on an information-theoretic argument: a polynomial number of queries, each of polynomial size, is not enough to identify each concept in the class, even if the learner has unlimited computation power between the queries. Hence, having approximate fingerprints implies non-learnability, regardless on any complexity-theoretic consideration.

The main result in this paper is that approximate fingerprints are also a necessary condition for proving non-learnability, at least for any proof relying on the amount of information received by the learner.

We take first a slight variation of Angluin's "approximate fingerprints", mainly to ensure that it is decidable whether a given object is an approximate fingerprint. It is easy to verify that this variation still allows to prove Theorem 1, and that the classes shown to have approximate fingerprints in [3] still have them with the new definition.

Then we show that, in any class *not* having approximate fingerprints, it is possible to implement a very weak version of majority vote, which leads to a polynomial learning strategy. More precisely, we restrict ourselves to "honest" classes, satisfying certain natural conditions specified in Section 2, and show the following fact.

Theorem 2 *For every honest class $\mathcal{C}$, if $\mathcal{C}$ does not have approximate fingerprints then it is learnable from Equivalence queries, given access to an oracle in the polynomial-time hierarchy.*

Furthermore, if we are happy with (not necessarily computable) identification strategies, we can remove the honesty condition and use Angluin's original definition.

Putting together Theorems 1 and 2, we obtain the following corollary.

Corollary 3 *If* P = NP*, every honest class* $\mathcal{C}$ *is learnable from Equivalence queries if and only if it does not have approximate fingerprints.*

Hence, it is still conceivable that classes without approximate fingerprints are learnable in polynomial time. More realistically, this corollary can be interpreted as follows: with the current knowledge, the only feasible way to prove that a class is not learnable is to show that it has approximate fingerprints; any other way would immediately yield a proof that P $\neq$ NP.

Finally, we apply the techniques in the proof of Theorem 2 to the class of boolean circuits.

Theorem 4 *If* P = NP *then boolean circuits are learnable from Equivalence queries.*

Thus, boolean circuits are different in a sense from the classes discussed in [3]. For those classes, negative results are absolute; for circuits, any negative result needs to assume or prove that P $\neq$ NP. In other words, the hardness of learning circuits relies on the intractability of computational problems, not on the amount of information needed for identification.

It would be interesting to delimit more precisely the frontier between "information" and "computation" hardness, especially when other types of queries are present. Let us remind that this problem is completely solved for the PAC-learning paradigm. It is known [5] that every honest class is PAC-learnable from a polynomial number of examples, given access to an oracle in NP. Hence, for PAC, all the difficulty comes from the computation side.

The paper is organized as follows: In Section 2 we establish some notation and recall the basic concepts of query learning. In Section 3 we define approximate fingerprints. Section 4 proves our main result, Theorem 2. Section 5 presents the application to circuits.

2 Preliminaries

Languages. All languages defined in this paper are subsets of $\Sigma^\star$, where $\Sigma = \{0,1\}$. For a string $x \in \Sigma^\star$, $|x|$ is the length of x. By $\Sigma^{\leq n}$ we denote the set $\{\ x \in \Sigma^\star\ :\ |x| \leq n\ \}$.

We use implicit and easily computable isomorphisms between natural numbers, strings, and tuples of strings. All polynomials we use are assumed to be nondecreasing.

We expect some familiarity with the central notions of complexity theory. We use in particular the classes P, NP, #P, and the $\Sigma_k^{\rm p}$ and $\Delta_k^{\rm p}$ classes of the polynomial-time hierarchy, PH. For two complexity classes $\mathcal{C}$ and $\mathcal{D}$, $\mathcal{C}(\mathcal{D})$ denotes the complexity class $\mathcal{C}$ relativized to class $\mathcal{D}$, i.e., the set of languages accepted by machines of type $\mathcal{C}$ relative to oracles in $\mathcal{D}$. See [4] for explanations.

Representation classes. We specify learning problems by means of representation classes, in the style of [11, 13]. A *representation class* $\mathcal{C}$ is a tuple (R, Φ), where

- language $R \subseteq \Sigma^\star$ is the set of *representations,* and
- function Φ maps each representation $r \in R$ to a language, or *concept,* $\Phi(r) \subseteq \Sigma^\star$.

If $r \in R$ and $\Phi(r) = c$ we say that r is a representation of concept c. We talk often about the *size* of a concept; we mean the length of a shortest representation for that concept.

We only deal with *honest* representation classes, those that satisfy the following two conditions: 1) it is easy to decide whether a given string is a representation, that is, R is in P; 2) it is easy to decide whether a string is in a given concept, that is, the set $\{\ (r,w)\ :\ w \in \Phi(r)\ \}$ is in P.

In particular, we will use a representation class of boolean circuits, *CIR.* It consists of the set $R_{\rm cir}$ of encodings of circuits and the function $\Phi_{\rm cir}$ that maps the encoding of circuit c to the set of bit strings on which c evaluates to 1.

Learning. We consider learning algorithms, or learners, that make Equivalence queries to a teacher. The teacher answers the queries according to a *target concept* c, taken from a representation class $\mathcal{C} = (R, \Phi)$. The target is

fixed for an execution of the learner. More precisely, a query is a representation $r \in R$, and the answer is either YES if $\Phi(r) = c$, or else any string in $\Phi(r) \Delta c$, called the *counterexample.*

The running time of the algorithm is measured as a function of the size of the target concept and the length of the longest counterexample received. The learner learns $\mathcal{C}$ if, for any target concept c taken from $\mathcal{C}$ and any sequence of answers consistent with c, it outputs a representation in R for c, and it does so in polynomial time.

3 Approximate Fingerprints Revisited

We recall the following definitions from Angluin's work [3]. All of them assume that $\mathcal{C} = (R, \Phi)$ is a honest representation class.

Definition 5 i) A class of concepts $C \subseteq \mathcal{C}$ is bounded by $m \in \mathbb{N}$ if for each $c \in C$ there is a representation $r \in R$ such that $\Phi(r) = c$ and $|r| \leq m$.

ii) A sequence of concept classes in $\mathcal{C}$, $\{C_n\}$, is a sequence C_1, C_2, C_3 ... such that each C_n is a class of concepts. Such sequence is *bounded by function* f if each C_n is bounded by $f(n)$.

iii) For a language A, $A^{\leq n}$ is the set $A \cap \Sigma^{\leq n}$; note that $A^{\leq n}$ is not necessarily a concept in $\mathcal{C}$, even if A is. For a class of concepts $\mathcal{A}$, $\mathcal{A}^{\leq n}$ is the class $\{\ A^{\leq n}\ :\ A \in \mathcal{A}\ \}$.

iv) Let T be a class of concepts, $w \in \Sigma^\star$ a string, b a value in $\{0, 1\}$, and α a real number between 0 and 1. We say that (w, b) is an *α-approximate fingerprint for* T if

$$\|\{\ c \in T\ :\ \chi_c(w) = b\ \}\| < \alpha \cdot \|T\|.$$

Definition 6 Representation class $\mathcal{C}$ *has approximate fingerprints* if

there exist polynomials p_1 and p_2 such that
  for every polynomial q
(1) there is a sequence of concept classes $\{T_n\}$ bounded by $p_1(n)$ and
(2) infinitely many n such that
(3) $T_n^{\leq p_2(n)}$ contains at least two sets, and
  for every $r \in R$ of length at most $q(n)$
    there is some $w \in \Sigma^{\leq p_2(n)}$ such that

(4) $(w, \chi_{\Phi(r)}(w))$ is a $1/q(n)$-
approximate fingerprint for $T_n^{\leq p_2(n)}$.

This definition is different from that in [3] in two respects. First, line (2) reads "for all sufficiently large n" in [3]. It is easy to verify that the weaker condition "for infinitely many n" is enough for the proof of Theorem 1. Intuitively, to prove non-learnability it is enough to force superpolynomial running time in each algorithm at infinitely many lengths.

Second, in lines (3) and (4) we use $T_n^{\leq p_2(n)}$ instead of the original T_n. Again, Angluin's proof goes through with this change, and the approximate fingerprints she finds for dfa, nfa, cfg, and CNF and DNF formulas also witness this property. The reason to introduce this change is the following: with the original definition, the property given by line (4) is not recursive in general, because it is not even decidable whether two representations define the same concept. The polynomial bound on the lengths involved makes this predicate recursive and puts it "close enough" to PH for our purposes.

Finally, note that we can replace lines (1) and (2) in Definition 6 by the equivalent

(1) there are infinitely many n such that
(2) there is some concept class T_n bounded by $p_1(n)$ such that,

that are slightly easier to handle.

4 Necessity of Approximate Fingerprints

In this section we prove the main result of the paper, Theorem 2 stated in the Introduction.

Suppose $\mathcal{C} = (R, \Phi)$ does not have approximate fingerprints. This means that

for every polynomials p_1 and p_2
there is some polynomial q such that
for all but finitely many n and
for every concept class T_n bounded by $p_1(n)$,
either $T_n^{\leq p_2(n)}$ contains less than two sets, or
for some $r \in R$ of length at most $q(n)$
and for every $w \in \Sigma^{\leq p_2(n)}$

$(w, \chi_{\Phi(r)}(w))$ is not a $1/q(n)$-
approximate fingerprint for $T_n^{\leq p_2(n)}$.

Notation. In order to describe the learning algorithm, let us introduce some notation.

A *sample* is a set of pairs of the form (w, b), where w is a string and $b \in \{0, 1\}$. We say that *concept c is consistent with sample s* when, for each pair (w, b) in s, $\chi_c(w) = b$.

The set $Cons(n, s)$ is the class of all concepts that have size at most n and are consistent with s. Note that, given $r \in R$, a sample s, and string 0^n, it is possible to decide in polynomial time whether $\Phi(r) \in Cons(n, s)$.

For a concept c and any n, we say that "n is large enough for c" if

- c has size at most n, i.e., some representation of size n or less, and
- every other concept c' of size at most that of c differs from c in at least one string of length $\leq n$.

Algorithm. We first describe an algorithm A for $\mathcal{C}$ that reads as input a natural number n, and has the following properties:

P1: Whenever A outputs a representation in R, it is a good representation for the target.

P2: If n is large enough for the target, $A(n)$ always outputs a representation in R.

P3: For every n, $A(n)$ always halts, and it does so in time polynomial in n and the length of the longest counterexample received.

Later we will remove the need for input n.

Fix $p_1(n) = p_2(n) = n$ and let q be the polynomial provided for p_1 and p_2 by the assumption that $\mathcal{C}$ does not have approximate fingerprints. Learner A is defined in Figure 1.

Invariant. The following two predicates are invariants of the main loop in A:

```
1.  input n;
2.  s := ∅;      { s is a sample };
3.  loop
4.      case ‖Cons(n, s)^{≤n}‖ = 0 :
5.          output "n not large enough" and halt;
6.      case ‖Cons(n, s)^{≤n}‖ = 1 :
7.          find the smallest r ∈ R such that Φ(r) ∈ Cons(n, s);
8.          ask r as a query;
9.          if answer = YES then
10.             output r and halt
11.         else
12.             output "n not large enough" and halt
13.         endif;
14.     case ‖Cons(n, s)^{≤n}‖ ≥ 2 :
15.         find some r ∈ R of length at most q(n) such that, for every w ∈ Σ^{≤n},
16.           (w, χ_{Φ(r)}(w)) is not a 1/(16q(n))-approximate fingerprint
17.           for Cons(n, s)^{≤n};
18.         ask r as query;
19.         if answer = YES then
20.             output r and halt
21.         else
22.             let w be the counterexample;
23.             s := s ∪ {(w, 1 − χ_{Φ(r)}(w))}
24.         endif
25. endloop
```

Figure 1. Learning algorithm A.

I1: If n is large enough for the target, then the target is in $Cons(n, s)$.

The predicate is initially true because every concept is consistent with the empty sample. And each pair added to s is consistent with the target, so *I1* is maintained.

I2: $\|Cons(n, s)^{\leq n}\| < 2^{n+1} \cdot \left(1 - \frac{1}{16q(n)}\right)^{\|s\|}$.

This is initially true because there are never more concepts in $Cons(n, s)$ than representations of size at most n, i.e., less than 2^{n+1}. At each iteration, at most one pair is added to s, but by the way r is chosen in line 15, this pair is consistent with at most a fraction $1 - \frac{1}{16q(n)}$ of sets in $Cons(n, s)^{\leq n}$. Hence *I2* is maintained.

Note that the r required in lines 15–17 always exists by the assumption that $\mathcal{C}$ does not have approximate fingerprints. The additional factor $1/16$ is introduced in order to find r with a PH oracle, as discussed later.

Termination. Note that when $\|Cons(n, s)^{\leq n}\| \leq 1$, A necessarily halts at this iteration. Furthermore, each iteration either halts or adds some element to s. By invariant *I2*, this can be repeated at most $i(n)$ times, where $i(n)$ is the maximum such that $2^{n+1} \cdot \left(1 - \frac{1}{16q(n)}\right)^{i(n)} > 1$. By substitution, it is easy to verify that $i(n) < \lceil 16(n+1)q(n)/\ln(2) \rceil$. Thus, the main loop is executed only a number of times polynomial in n.

Correctness. Note that whenever A outputs a representation, that representation has been just answered YES, so it is correct with respect to the target. This shows property *P1* of A.

The learner can halt without giving a representation in two points, lines 5 and 12. In line 5, $Cons(n, s)$ is empty. By invariant *I1*, this means that n is not large enough. In line 12, two things can happen: either n is less than the size of the target, so it is not large enough; or else, concept $\Phi(r)$ has size at most that of the target, $\Phi(r) \neq$ target, but $\Phi(r)$ and the target agree on all strings of length at most n. Because r is chosen smallest in line 7, again n is not large enough. This shows property *P2*.

Time complexity. There are four steps of A that require querying an oracle.

Line 4: Clearly, $Cons(n,s)^{\leq n}$ is empty if and only if $Cons(n,s)$ is. This is equivalent to saying that for every $r \in R$ of size at most n, $\Phi(r) \notin Cons(n,s)$. Predicate $\Phi(r) \notin Cons(n,s)$ can be verified in polynomial time, as said before, so this is a coNP question; it can be solved by a polynomial-size query to an NP oracle.

Line 6: Knowing whether $\|Cons(n,s)^{\leq n}\| \leq 1$ is knowing whether, for every r_1, r_2 of length at most n, $\Phi(r_1) \in Cons(n,s)$ and $\Phi(r_2) \in Cons(n,s)$ implies $\Phi(r_1)^{\leq n} = \Phi(r_2)^{\leq n}$. This is again a coNP query.

Line 7: Since "$\Phi(r) \in Cons(n,s)$" is a P predicate, this part can be solved querying an oracle in NP.

Lines 15 to 17: These lines can be implemented querying an oracle in Σ_6^{p}. In this version we only sketch the proof.

At this point of the algorithm, we know that $Cons(n,s)^{\leq n}$ contains at least two sets. By the assumption that $\mathcal{C}$ does not have approximate fingerprints, we know that there is some $r_0 \in R$ of length at most $q(n)$ such that for every $w \in \Sigma^{\leq n}$

$$\frac{\|\{\ c \in Cons(n,s)^{\leq n}\ :\ \chi_c(w) = \chi_{\Phi(r_0)}(w)\ \}\|}{\|Cons(n,s)^{\leq n}\|} \geq \frac{1}{q(n)}. \tag{1}$$

The function that, given 0^n and s returns $\|Cons(n,s)^{\leq n}\|$ can be shown to be in $\#\mathrm{P}(\Sigma_2^{\mathrm{p}})$, but it is not clear how to compute it in PH. We can, however, obtain good approximations of this function using a theorem by Stockmeyer ([12], Theorem 3.1) or, rather, its relativization in the presence of Σ_2^{p} oracles. This theorem guarantees that there is some function $f \in \Delta_3^{\mathrm{p}}(\Sigma_2^{\mathrm{p}}) = \Delta_5^{\mathrm{p}}$ such that for every n and s,

$$\frac{1}{2}\, f(0^n, s) \leq \|Cons(n,s)^{\leq n}\| \leq 2\, f(0^n, s).$$

A function $g(0^n, s, r, w)$ with similar properties can be obtained for the numerator of the left-hand side of equation (1).

Now, lines 15 to 17 are implemented searching for any r such that for all $w \in \Sigma^{\leq n}$ it holds

$$\frac{g(0^n, s, r, w)}{f(0^n, s)} \geq \frac{1}{4q(n)}.$$

The search for r uses an oracle in $\Sigma_2^{\mathrm{p}}(\Delta_5^{\mathrm{p}}) = \Sigma_6^{\mathrm{p}}$; here, Σ_2^{p} represents the guessing of r and the universal verification of w, and Δ_5^{p} is used to compute f and g.

By the way f and g are defined, the quotient g/f differs by a factor or at most 4 from the left-hand side of equation (1). Hence, at least representation r_0 satisfies this condition and r is always found in line 7. By a similar argument, any $r \in R$ satisfying this condition also satisfies

$$\frac{\|\{\ c \in \mathit{Cons}(n,s)^{\leq n}\ :\ \chi_c(w) = \chi_{\Phi(r_0)}(w)\ \}\|}{\|\mathit{Cons}(n,s)^{\leq n}\|} \geq \frac{1}{16q(n)},$$

as required by the definition of algorithm A.

Let us note that a more careful use of Stockmeyer's techniques, along the lines of [8, 9], brings the required oracle down to Σ_3^{p}.

If these lines are solved as explained, clearly each iteration of the loop takes time polynomial in n and the maximum length of a counterexample received so far. As the number of iterations bounded by a polynomial in n, property *P3* holds for A.

Getting rid of input n. To obtain the claimed learning algorithm for $\mathcal{C}$, it is enough to execute $A(i)$ sequentially with inputs $i = 1, 2, 3, 4, \ldots$ When some $A(i)$ outputs some $r \in R$ as output, output r and halt.

Correctness is clear because any representation output by any $A(i)$ must be correct (property *P1*). For termination, note that each $A(i)$ is terminating, and that $A(i)$ will output a representation whenever i is large enough for the target concept (properties *P3* and *P2*).

To discuss the time complexity, let N be the size of the target concept, n the minimum such that $A(n)$ outputs a representation, and l_i the length of the longest counterexample received by $A(i)$. There are two cases:

Case 1: $n \leq N$. Each run of $A(i)$ with $i \leq n$ takes time polynomial in i and l_i, i.e., polynomial in N and $\max_i\{l_i\}$.

Case 2: $n > N$. Note that the first **case** of algorithm $A(i)$ can occur only when $i < N$. So for all i with $N \leq i < n$, $A(i)$ must terminate in the second **case**. That is, $A(i)$ finds some r such that $\Phi(r)^{\leq i} = \mathrm{target}^{\leq i}$, but $\Phi(r) \neq \mathrm{target}$. Hence, the counterexample received in line 8 has length greater than i. The running time of $A(i)$ for all such i is polynomial in i and $l_i > i$, i.e., polynomial in $\max_i\{l_i\}$.

In summary, each run $A(i)$ takes time polynomial in N and $\max_i\{l_i\}$, and there are at most $\max_i\{N, \max_i\{l_i\}\}$ runs. So the new learner runs in polynomial time.

5 On the hardness of learning boolean circuits

As said in the Introduction, we can prove the following theorem.

Theorem 4 If P = NP then *CIR* is learnable from Equivalence queries.

A way to prove Theorem 4 would be to show that the class *CIR* does not have approximate fingerprints. Then this corollary would follow from Theorem 2.

Unfortunately, we have been unable to prove this, even assuming P = NP. A first problem is that each circuit accepts a set of strings of a fixed length, and this trivially prevents the negation of the approximate fingerprint definition. A deeper problem is that the number of possible concept classes T_n in the definition is doubly exponential, while the number of circuits of size $q(n)$ grows only exponentially.

Nevertheless, looking into the proof of Theorem 2, we notice that the hypothesis that $\mathcal{C}$ does not have approximate fingerprints is not applied to any arbitrary class T_n of concepts. It is applied only to classes of the form $Cons(n, s)$, each of which is described completely by a sample of polynomial size.

Using this fact, it is possible to give an ad-hoc version of algorithm A in Section 4 that works for the class *CIR*. We omit its presentation in this version; a complete proof can be found in [7].

Late addition. Note that our proof of Theorem 4 *does not* show that *CIR* is learnable with the help of an oracle in PH; it uses the full force of the assumption that P = NP. Very recently, Bshouty, Cleve, and Tamon have independently shown that *CIR* is learnable both by randomized algorithms with an NP oracle and by deterministic algorithms with a Σ_3^{p} oracle [6], thus obtaining Theorem 4 as a corollary. Many similar results concerning other representation classes are also shown in [6].

Acknowledgements. I am grateful to David Guijarro and Vijay Raghavan for helpful comments. I also thank Nader Bshouty, Richard Cleve, and

Christino Tamon for promptly sending a copy of their work [6], and Vijay again for first telling me about it.

References

1. D. Angluin: "Learning regular sets from queries and counterexamples". *Information and Computation* **75** (1987), 87–106.

2. D. Angluin: "Queries and concept learning". *Machine Learning* **2** (1988), 319–342.

3. D. Angluin: "Negative results for equivalence queries". *Machine Learning* **5** (1990), 121–150.

4. J.L. Balcázar, J. Díaz, and J. Gabarró: *Structural Complexity I.* EATCS Monographs on Theoretical Computer Science, vol. 11. Springer-Verlag 1988.

5. A. Blumer, A. Ehrenfeucht, D. Haussler, and M.K. Warmuth: "Occam's razor". *Information Processing Letters* **24** (1987), 377–380.

6. N.H. Bshouty, R. Cleve, S. Kannan, and C. Tamon: "Oracles and queries that are sufficient for exact learning". To be presented at the 7th COLT Conference, New Jersey, July 1994.

7. R. Gavaldà: *Kolmogorov Randomness and its Applications to Structural Complexity Theory.* Doctoral Dissertation, Universitat Politècnica de Catalunya, april 1992.

8. R. Gavaldà: "Bounding the complexity of advice functions". *Proceedings of the 7th Annual Conference on Structure in Complexity Theory,* 249–254. IEEE Computer Society Press, 1992.

9. J. Köbler: "Locating P/poly optimally in the extended low hierarchy". *Proceedings of the 10th Symposium on Theoretical Aspects of Computer Science,* 28–37. Lecture Notes in Computer Science 665. Springer-Verlag, 1993.

10. W. Maass and G. Turán: "On the complexity of learning from counterexamples". *Proceedings of the 30th Annual Symposium on Foundations of Computer Science,* 262–267. IEEE Computer Society Press, 1989.

11. M.K. Warmuth: "Towards representation independence in PAC learning". *Proceedings of the International Workshop on Analogical and Inductive Inference AII-89,* 78–103. Lecture Notes in Artificial Intelligence 397. Springer-Verlag, 1989.

12. L. Stockmeyer: "On approximation algorithms for #P". *SIAM Journal on Computing* **14** (1985), 849–861.

13. O. Watanabe: "A formal study of learning via queries". *Proceedings of the 17th International Colloquium on Automata, Languages, and Programming,* 139–152. Lecture Notes in Computer Science 443. Springer-Verlag, 1990.

14. T. Yokomori: "Polynomial-time learning of very simple grammars from positive data". *Proceedings of the 4th ACM Workshop on Computational Learning Theory,* 213–227. Morgan Kaufmann, 1991.