A framework for polynomial time query learnability

Osamu Watanabe

Report LSI–90–31

A Framework for Polynomial Time Query Learnability*

Osamu Watanabe
Dept.of Computer Science, Tokyo Institute of Technology, Tokyo 152, Japan
watanabe@cs.titech.ac.jp

July 7th, 1990

1. Introduction

The purpose of this paper is to prepare a formal framework for studying "polynomial time query learnability". Suppose someone, who is called a *teacher*, has some set, which is called a (*target*) *concept*, in his mind. The learning notion considered in this paper is to obtain the description of the concept by asking queries on it to the teacher. This type of learning is called *learning via queries* or *query learning*. In particular, we are interested in *polynomial time query learning*, i.e., query learning that terminates within polynomial time.

Angluin [Ang87] showed a learning algorithm for regular sets, where deterministic finite state automata (in short, dfa) are used for describing regular sets. The learning achieved by this algorithm is a typical example for polynomial time query learning. For any regular set X, Angluin's algorithm S_A asks queries about X and obtains the description of X, i.e., a dfa that accepts X. Queries asked by S_A are one of the following types:

(1) A *membership query* w; that is, S_A presents a string w to the teacher and asks whether w is in X.

*This is an extended version of a part of [Wa90]. The preparation of this paper was done while the author was visiting Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, and supported in part by ESPRIT II Basic Research Actions Program of the EC under contract No. 3075 (project ALCOM) and by Grant in Aid for Scientific Research of the Ministry of Education, Science and Culture of Japan under Grant-in-Aid for Co-operative Research (A) 02302047 (1990).

(2) An *equivalence query* M; that is, S_A presents a dfa M to the teacher and asks whether $L(M)$ (i.e., the set accepted by M) is equivalent to X. For this type of query, S_A expects a *counterexample* to the equivalence conjecture (i.e., a string in $(L(M)-X)\cup(X-L(M))$) if $L(M)$ is not equivalent to X. (Thus, this type of query is more precisely called an *equivalence query requiring a counterexample.*)

For any regular set X, S_A obtains the minimum state dfa M_X for X if each query is answered correctly w.r.t. X. Furthermore, it outputs M_X within polynomial time w.r.t. the number of states in M_X and the length of the longest counterexample. Thus, we can consider that the class of regular sets (or, more precisely, the class of regular sets represented by dfa) is polynomial time learnable by using membership and equivalence queries. It has been asked [Ang87, Ang88, BR87, Ish89, Sak88] whether similar learnability results hold for the other concept classes such as the class of regular sets represented by regular expressions, the class of context-free languages represented by context-free grammars, etc. This paper provides a formal framework for investigating this type of polynomial time learnability.

There are several learning models, and for some of them, a reasonable framework has been already established that provides notations and notions necessary for our discussions. However, those frameworks are not suitable for our type of learning. In what follows, we review major learning notions and point out the difference from ours.

Gold [Gol67] introduced a learning notion that is usually called the *identification in the limit* or the *EX identification*. For this learning notion, we have a framework that is commonly used now (see, e.g., [AS83]). Furthermore, such a framework has been extended [GS89] in order for studying "query learning". However, in this traditional type of learning, the main issue is the learnability within finite amount of time and is not the learnability within a given amount of time. In other words, the framework is for recursion theoretical learning and is not suitable for resource bounded learning.

On the other hand, the *PAC learning* introduced by Valiant [Val84] concerns resource bounded, in particular, polynomial time bounded learning. Due to the fundamental works in this field [BEHW86, PV88, PW88, War89], we have now what could be called a "framework" for the PAC learning (see, e.g., [PW88]). However, the learning notion considered in that framework is "passive learning" because learning algorithms are regarded as functions that compute the description of a target concept from a set of examples that are given as an input data. In other words, such a framework does not consider the situation where learners can ask questions about a target concept. (Note that the philosophy of "Probably Approximately Correct (PAC)" itself does not contradict to a query learning

model at all. Indeed, Valiant considered "query learning" briefly in [Val84]. However, the present framework for the PAC learning seems too restrictive for discussing query learning; see also Section 4.3.)

In this paper, we prepare notations and clarify notions that are necessary for discussing the time bounded version of query learning. We will consider the following points:

(1) How can we specify a target concept class, and what kind of concept classes are reasonable?

(2) How can we describe and analyze our learning algorithms?

(3) What is an appropriate definition for the goal of learning? In other words, in which situation can we claim appropriately that 'a target concept has been learned'?

The first two points are mainly matters of notations, which will be settled by choosing and/or introducing appropriate ones. On the other hand, the third point concerns the definition of learning notion itself. Here we will propose three definitions each of which defines a different learning notion. Some examples will be given to motivate those definitions.

Preliminaries

In this paper we follow standard definitions and notations in computational complexity theory; the reader will find them in a standard text book such as [HU79, BDG88].

Throughout this paper, we use a fixed alphabet Σ that contains $\{0,1\}$. A symbol $\perp \notin \Sigma$ is used to denote "undefined". By a *string* we mean element of Σ^*, and by a *language* we mean a subset of Σ^*. We use ϵ to denote the null string. For any set A, let $\overline{A}$ to denote the complement of A, i.e., $\Sigma^* - A$; for any set A and B, let $A \triangle B$ denote the set $(A - B) \cup (B - A)$. The length of a string x is denoted by $|x|$. Let $A^{\leq n}$ is used to denote the set $\{x \in A : |x| \leq n\}$.

We use some reasonable encoding schema over Σ for several objects; e.g., nonnegative integers, deterministic/nondeterministic finite automata, regular expressions, right-linear grammars, context-free grammars, Turing machines, boolean circuits, etc. We can assume, e.g., that the length of the string encoding a dfa M with input alphabet Σ is polynomially related to the number of states in M. In the following, we will not distinguish an object and the string encoding it. For example, we use $\mathbf{N}$ to denote both the set of nonnegative integers and the set of their binary notations. We consider the following pairing function $\lambda xy.\langle x,y\rangle$: for any string x and y, $\langle x,y\rangle$ is a string of length $2(|x| + |y| + 1)$ from which one can decode both x and y in linear time. For each $n > 2$, this pairing function can be easily extended to an n-tuple function $\lambda x_1, ..., x_n.\langle x_1, ..., x_n\rangle$.

2. Representation Class

In this section the framework concerning "target concept" is discussed. We first determine our way to specify a target concept class. We adopt the way that has been used in the PAC learning [PW88, War89], and specify a concept class in terms of "representation class". There are representation classes whose polynomial time non-learnability is almost trivial from their definition. In order to avoid *some of* such ill-natured cases, we propose conditions for representation classes.

We should notice that the polynomial time learnability of a given concept class may largely depend on a way of describing the concepts. For example, we know by Angluin's algorithm that the class of regular sets represented by dfa is polynomial time query learnable; on the other hand, it is still open whether the same learnability result holds for the class of regular sets represented by regular expressions. Thus, the learning problem is not well specified until we determine both a concept class and a way of representing the concepts.

This point has been already noticed in the PAC learning. Indeed, it is observed [PW88] that the descriptive power of representation language is a significant factor for the existence of an efficient PAC learning algorithm. Motivated this observation, researchers have started specifying concept classes more precisely by using "representation class" [PW88, War89]. We will follow this idea and adopt almost the same notation for our framework.

A *representation class* is a five-tuple $(R, \Phi, \rho, \Gamma_1, \Gamma_2)$, where

- Γ_1 and Γ_2 : finite sets, called an *alphabet for representations* and an *alphabet for concepts* respectively,
- R : a subset of Γ_1^*, called a *representation language*,
- Φ : a function from R into Γ_2^*, called a *semantic function* or *concept mapping*, and
- ρ : a function from R to $\mathbf{N}$, called a *size function*.

(Note that the notation differs slightly from [PW88, War89]. In [PW88] a representation class was a pair, in [War89] it became a four-tuple, and now it is defined by a five-tuple! But do not worry; the last three components are, as we will soon see, usually omitted.)

For each $r \in R$, the set $\Phi(r)$ is called a *(target) concept*. And the class of concepts $\{\Phi(r) : r \in R\}$ is called a *concept class*. Since we are using the fixed alphabet Σ, we will omit specifying alphabets Γ_1 and Γ_2. A size function ρ is also omitted when it is the length function, i.e., $\lambda x.|x|$. Thus, a representation class is often written as (R, Φ).

Although we are not really ready to define the notion of "polynomial time learning

algorithm" formally, it might be better to give a rough explanation on this notion at this point. Let $C = \langle R, \Phi, \rho \rangle$ be any representation class. A *learning algorithm* for C is an algorithm S such that for every concept X (or, more precisely, every $r \in R$), S obtains a representation r' for X $(= \Phi(r))$ by asking queries on X. Note that the output r' is not necessarily the same as r. We say that S is *polynomial time* if for every $r \in R$, its running time for learning $\Phi(r)$ on input $\rho(r)$ is polynomially bounded by some polynomial in $\rho(r)$. (Note that these definitions are still not precise; the formal ones will be given at Section 4.)

Now we present some examples of representation classes. In the following examples, we often omit explaining semantic functions when they are clear from the context.

Example 2.1. First consider representation classes from formal language theory.

- $DFA = (R_{\mathrm{dfa}}, \Phi_{\mathrm{dfa}}, \rho_{\mathrm{dfa}})$, where R_{dfa} is the set of dfa (encoded in Σ^*), and for any $r \in R_{\mathrm{dfa}}$, $\Phi_{\mathrm{dfa}}(r)$ is the set accepted by dfa (encoded by) r and $\rho_{\mathrm{dfa}}(r)$ is the number of states in the dfa (encoded by) r.
- $NFA = (R_{\mathrm{nfa}}, \Phi_{\mathrm{nfa}}, \rho_{\mathrm{nfa}})$, where R_{nfa} is the set of nondeterministic finite automata, and for any $r \in R_{\mathrm{nfa}}$, $\rho_{\mathrm{nfa}}(r)$ is the number of states in r.
- $REG = (R_{\mathrm{reg}}, \Phi_{\mathrm{reg}})$, where R_{reg} is the set of regular expressions,
- $RG = (R_{\mathrm{rg}}, \Phi_{\mathrm{rg}})$, where R_{rg} is the set of right-linear grammars,
- $CFG = (R_{\mathrm{cfg}}, \Phi_{\mathrm{cfg}})$, where R_{cfg} is the set of context-free grammars written in Chomsky normal form,
- $REG\text{-}CFG = \langle R_{\mathrm{regcfg}}, \Phi_{\mathrm{cfg}} \rangle$, where $R_{\mathrm{regcfg}} \subseteq R_{\mathrm{cfg}}$ is the set of context-free grammars that generate regular sets, and
- $PTM = (R_{\mathrm{ptm}}, \Phi_{\mathrm{ptm}})$, where R_{ptm} is the set of polynomial time Turing machines.

Now Angluin's algorithm S_A is explained more precisely by saying that 'S_A is a learning algorithm for *DFA*.' The existence of a similar learning algorithm for the other representation classes except *PTM* has been left open. (For *PTM*, we can prove that such a learning algorithm does not exist; see the later discussion of this section.) Notice that one can easily modify a learning algorithm for *NFA* to one for *RG*, and vice versa. Again such an equivalence relation has not been known for the others.

Example 2.2. We show some examples concerning geometrical concepts. We define representation classes for "ball" and "polygon" in $[0,1]^k$.

Basically, we represent a ball by its center and its radius, and represent a polygon by its vertices. For these purposes, we need to encode points in $[0,1]^k$ by Σ^*. Here we use the following simple encoding schema.

Let us first consider $[0,1]$. Clearly, all the points cannot be encoded into Σ^*; that is, every coding schema works only on some countable subset of $[0,1]$. Our coding schema works for the set of numbers in $[0,1)$ whose binary representation is finite; let $\mathbf{B}[0,1]$ denote it, and in general, let $\mathbf{B}[a,b]$ denote the set $[a,b] \cap \mathbf{B}[0,1]$. We encode $\mathbf{B}[0,1]$ over $\{0,1\}$ as follows: for every $x \in \mathbf{B}[0,1]$, the code of x is the shortest $w \in \{0,1\}^*$ such that $0.w$ is a binary representation of x. (Note that the null string encodes the number 0. in $\mathbf{B}[0,1]$.) By using our tupling function, we can extend this encoding schema to $\mathbf{B}[0,1]^k$, the set of points in $[0,1)^k$ whose coordinates have finite binary representations. In the following, we identify a point in $\mathbf{B}[0,1]^k$ and its code.

Now define the following representation classes:

- 2-$BALL = \langle R_{\text{2-ball}}, \Phi_{\text{2-ball}} \rangle$,
  where · $R_{\text{2-ball}} = \{\langle u, d \rangle : u \in \mathbf{B}[0,1]^2 \text{ and } d \in \mathbf{B}[0,1]\}$, and
  · for any $\langle u, d \rangle \in R_{\text{2-ball}}$, $\Phi_{\text{2-ball}}(\langle u, d \rangle) = \{v \in \mathbf{B}[0,1]^2 : d(u,v) \leq d\}$.
  (Where $d(u,v)$ is the distance between u and v.)
- 2-$POLYGON = \langle R_{\text{2-polygon}}, \Phi_{\text{2-polygon}} \rangle$,
  where · $R_{\text{2-polygon}} = \{\langle u_1, ..., u_l \rangle : u_1, ..., u_l \in \mathbf{B}[0,1]^2\}$, and
  · for any $\langle u_1, ..., u_l \rangle \in R_{\text{2-polygon}}$, $\Phi_{\text{2-polygon}}(\langle u_1, ..., u_l \rangle) = P(u_1, ..., u_l) \cap \mathbf{B}[0,1]^2$.
  (Where $P(u_1, ..., u_l)$ is the polygon determined by vertices u_1, ..., u_l.)

The concepts specified by these representation classes are regarded as "ball" and "polygon" in $\mathbf{B}[0,1]^2$ respectively. For any $k \geq 1$, representation classes k-$BALL$ and k-$POLYGON$, which respectively corresponds k-dimensional balls and k-dimensional polygon, are defined similarly.

Example 2.3. A *boolean circuit* or a *logical circuit* is an acyclic circuit consisting of gates that achieve basic boolean operations such as $\neg$, $\wedge$, and $\vee$. We consider only boolean circuits with one output gate, and regard such circuits as acceptors. For any circuit C with n input gates and any a_1, a_2, ..., $a_n \in \{0,1\}$, let $C(a_1, ..., a_n)$ denote the output of C when a_1, a_2, ..., a_n are given to the 1st, 2nd, ..., nth input gates respectively; we say that C *accepts* a string $a_1 a_2 \cdots a_n$ $(\in \{0,1\}^n)$ if $C(a_1, ..., a_n) = 1$. Thus, we can use boolean circuits for representing sets of strings. Now we define a representation class as follows: $CIR = (R_{\text{cir}}, \Phi_{\text{cir}})$, where R_{cir} is the set of circuits, and for any circuit $r \in R_{\text{cir}}$, $\Phi_{\text{cir}}(r)$ is the set of strings accepted by r.

The main motivation of preparing our framework is to investigate "polynomial time query learnability" formally and to develop some methods to determine whether a given

concept class has a polynomial time learning algorithm. For this purpose, however, our definition of "representation class" may be too naive because one can define several strange representation classes for which polynomial time (non-)learnability is clear. Since we cannot obtain a good insight into "polynomial time learnability" through the investigation of such ill-natured representation classes, we had better exclude them from our consideration. Here we propose conditions for excluding some (but not all) such representation classes.

In the following, let $C = \langle R, \Phi, \rho \rangle$ denote any representation class.

Condition for Size Function

Even in the case that a size function is not the length function, it is natural to assume that the size of each representation r, $\rho(r)$, reflects its length, $|r|$. Indeed, in most cases we can assume the following condition:

$$\textbf{Condition for Size Func.:} \quad \exists p : \text{a polynomial}, \forall r \in R \; [\; |r| \leq p(\rho(r)) \;].$$

By this condition we can avoid such an extreme case that C has no polynomial time learning algorithm because some representations are too long compared with their size and thus are not even printable within polynomial time w.r.t. their size.

Condition for the Syntax of Representation Language

Recall the equivalence query used in Angluin's algorithm, i.e., the type of query that asks whether a query string r represents the same set as the concept. For this type of query, we usually assume that r is in R, i.e., a query string is a syntactically correct; otherwise, we may be able to abuse teacher's ability to answer to equivalence queries for deciding whether a given word is in R. One way to avoid such abusing (or to guarantee that a query string is always in R) is to assume that the syntax check for R is not difficult; thereby, a learner can check whether a query string is in R or not by itself. This is the motivation for the following condition.

$$\textbf{Condition for Rep. Class:} \quad R \in \mathrm{P}.$$

This condition may not be necessary for "subconcepts". For example, consider the representation class *REG-CFG* defined at Example 2.1. Clearly, R_{regcfg} is not in P; however, this representation class is still "reasonable" if we consider the problem of learning *REG-CFG* by using a teacher for *CFG*. That is, *REG-CFG* is, though it is not a reasonable representation class, still a reasonable representation subclass of *CFG*.

Condition for the Complexity of Concepts

Suppose that a representation class C can express concepts of a higher complexity class than P. It is reasonable to consider that such a representation class is too complicated to be learned by some polynomial time learning algorithm. Indeed, we can easily prove that if C can express $DTIME(2^{\mathrm{poly}})$ sets, then C is not polynomial time learnable. This fact motivates us to introduce a condition concerning the computational complexity of concepts.

Note that such a condition that every concept is in P (i.e., $\{\Phi(r) : r \in R\} \subseteq \mathrm{P}$) is still weak for our purpose. Take the representation class PTM for example. Note that PTM satisfies the condition; however, we can easily prove that PTM has no polynomial time learning algorithm. What follows is the idea of the proof: Define a sequence of sets $\{L_i\}_{i\geq 1}$ such that each L_i diagonalizes the ith polynomial time learner, i.e., L_i is designed so that it is not learnable by the ith polynomial time learning algorithm; thereby, we can conclude that no polynomial time learning algorithm exists for $\{L_i\}_{i\geq 1}$. Since any polynomial time algorithm can be diagonalized by some polynomial time algorithm, we can define $\{L_i\}_{i\geq 1}$ so that each L_i is recognized by some polynomial time Turing machine. That is, $\{L_i\}_{i\geq 1}$ $\subseteq \{\Phi_{\mathrm{ptm}}(r) : r \in R_{\mathrm{ptm}}\}$. Therefore, no polynomial time learning algorithm exists for the concept class $\{\Phi_{\mathrm{ptm}}(r) : r \in R_{\mathrm{ptm}}\}$. The construction of $\{L_i\}_{i\geq 1}$ has worked because the above condition requires a polynomial time upper bound for each concept respectively. This observation leads us to the following *uniform* upper bound condition.

Uniform P-Computability Condition:

$\exists M$: a polynomial time Turing machine,

$\forall r \in R, \forall u \in \Sigma^*$ [$u \in \Phi(r) \leftrightarrow M$ accepts $\langle r, u\rangle$].

A similar condition is considered in [War89].

We can easily show that each representation class defined at Example 2.1 (except PTM) satisfies this condition; those proofs are left to the reader. On the other hand, there are some interesting representation classes that do not satisfy this condition. For example, if P $\neq$ NP, then a representation class for "pattern language" [Ang80] does not satisfy this condition because the membership problem for some pattern language is NP-complete [Ang80]. Thus, the reader should be aware that the Uniform P-Computability Condition may exclude some interesting representation classes as well as uninteresting ones.

3. Learning System

We define our computational model for query learning protocols. In order to describe learning protocols, we introduce "learning system", which is similar to interactive proof systems [GMR85] (where "prover" and "verifier" in the interactive proof system are replaced by "teacher" and "learner" in the learning system).

Intuitively, a learning system is a pair of a teacher and a learner that can communicate each other during the execution of the system. Consider a learning system that is designed for learning some representation class $C = (R, \Phi)$. The goal of the system is to have the learner to obtain (i.e., output) a representation $r \in R$ of a target concept that is made known only to the teacher. Here we describe this idea more formally.

A *learning system* $\langle S, T \rangle$ is a pair of a Turing machine S and an oracle function T, where S is called a *learner* and T is called a *teacher*. Note that a teacher T is regarded not as a machine but as a function; that is, we are assuming that T can *somehow* answer to queries *immediately*. We take such a computation model because our interest is not "teaching strategy" but "learning strategy". (For the sake of simplicity, however, we will treat a teacher T as a machine in the following explanation.)

A learning system $\langle S, T \rangle$ is regarded as a pair of Turing machines that shares one work tape. On the other hand, some of those tapes are used for specific purposes. The following are the tapes used in $\langle S, T \rangle$:

a communication tape:
    a read-write work tape shared by S and T,
an target tape:
    a read-only tape for T and not accessible by S,
an input tape, an output tape, and work tapes:
    ordinary input, output, and work tapes for S not accessible by T.

Note that T has access only to the communication tape and the target tape. (Later, in Section 4.2, we will consider one variation such that an input tape is also accessible by a teacher.) The communication tape is used for the communication between S and T (see the explanation in the next paragraph). The role of the target tape is to let the teacher T know a target concept. Thus, a representation r of a target concept, a *target representation*, is usually written on the target tape; this situation intuitively means that T knows the concept that is represented by r. Teacher T who knows r (or, more precisely, T with r on the target tape) is written as "$T(r)$".

Let $\langle S, T\rangle$ be any learning system. Prior to the execution, an input x and a target representation r are given respectively on the input tape and the representation tape. Then the computation of $\langle S, T\rangle$ (which is written as $\langle S, T(r)\rangle(x)$) starts from S, executes S and T in turn, and finally halts at S. More specifically, S and T run as follows:

(1) S starts from its initial state, writes a query string on the communication tape, enters a query state, and stops;

(2) T writes an answer to the query asked at (1) on the communication tape, and stops;

(3) S resumes its execution, $\cdots$.

$\vdots$

(k) S resumes its execution, enters a final state, and halts.

If S outputs y on its output tape and halts with an accepting state at (k), we say that $\langle S, T(r)\rangle(x)$ *outputs* y (and write $\langle S, T(r)\rangle(x) = y$).

Obviously, the learning problem becomes trivial if learners can ask any queries (or teachers can answer to any queries). Thus, in the study of query learning, we put a strong restriction on the types queries and answers and discuss learnability under the restriction. Here we explain four query types and two answer types that are often considered in many different contexts. (In [Ang88] the reader will find a good survey on several query types that have been proposed in the literature.)

Suppose that we are learning some target concept X in the concept class represented by $C = (R, \Phi)$. We consider the following query types:

membership query (Mem): for a query string[1] v, the query asking whether $v \in X$,

equivalence query (Equ): for a query string r, the query asking whether $\Phi(r) = X$,

subset query (Sub): for a query string r, the query asking whether $\Phi(r) \subseteq X$, and

superset query (Sup): for a query string r, the query asking whether $\Phi(r) \supseteq X$.

For each query, an answers from a teacher is in one of the following:

yes/no answer: either 1 (meaning 'yes') or 0 (meaning 'no'), and

answer with a counterexample: either a string w in Σ^* (meaning 'w is a counterexample to the query') or $\perp$ (meaning 'no counterexample exists').

We consider only yes/no answers to membership queries.

Let $C = (R, \Phi)$ be any representation class. A teacher T is called *consistent* with C (or, more simply, a *teacher for* C) if for every $r \in R$, $T(r)$ provides correct answers w.r.t. $\Phi(r)$. We can state "consistency" more specifically by choosing query types and answer

[1] A teacher must be informed on the type of a query being asked; thus, precisely speaking, each query string should contain such information. However, by "query string" we often mean the string that is being asked.

types. For example, a teacher is called an *(Mem,Sub;Equ)-teacher consistent with C* (or, more simply, a *(Mem,Sub;Equ)-teacher for C*) if it replies yes/no answers to membership queries and subset queries, and answers with a counterexample to equivalence queries, and those answers are correct w.r.t. C. A tuple such as (Mem,Sub;Equ) is called a *teacher type*.

Example 3.1. It is easy to check that Angluin's learning algorithm S_A satisfies the following property:

(C.1) $\forall T$: a (Mem;Equ)-teacher for *DFA*, $\forall r \in R_{\mathrm{dfa}}$
$[\ \langle S_A, T(r)\rangle(x) = r_0\ ]$,
where r_0 is the minimum state dfa for $\Phi_{\mathrm{dfa}}(r)$.
(*NOTE:* Since S_A needs no input string, x can be any string in Σ^*.)

Intuitively, this property means that from every consistent (Mem;Equ)-teacher for *DFA*, S_A can obtain the minimum state dfa for a target regular set; or we can claim more simply that S_A can learn *DFA* from every (Mem;Equ)-teacher for *DFA*.

Finally, we introduce some notations for measuring the time complexity of a learning system. For any learning system $\langle S, T\rangle$, we define the following complexity measures:

for every input x and target representation r,
$\#q_S(x; T(r)) \stackrel{\mathrm{def}}{=}$ the number of counterexample queries asked by S
during the computation $\langle S, T(r)\rangle(x)$, and
for every input x, target representation r, and i, $0 \leq i \leq \#q_S(x; T(r))$,
$\#steps_S(x, i; T(r)) \stackrel{\mathrm{def}}{=}$ the number of steps spent by S
after it receives the answer to the ith counterexample query
until it asks the next counterexample query, or halts.

Since we are ignoring the computation by T, the system's execution time is measured by the following $time_S$:

for every input x and target representation r,
$time_S(x; T(r)) \stackrel{\mathrm{def}}{=} \sum_{i=0}^{\#q_S(x;T(r))} \#steps_S(x; T(r))$.

Example 3.2. The following property for Angluin's algorithm S_A is easily verified:

(C.2) $\exists p, q$: polynomials,
$\forall T$: a (Mem;Equ)-teacher for DFA, $\forall r \in R_{\mathrm{dfa}}$
[(i) $\#q_{S_A}(x; T(r)) \leq p(n_0)$, and
(ii) $\forall i, 0 \leq i \leq \#q_{S_A}(x; T(r))$ [$\#step_{S_A}(x, i; T(r)) \leq q(i + l_i)$]],
where n_0 is the number of states in the minimum state dfa for $\Phi_{\mathrm{dfa}}(r)$, and
l_i is the length of the answer to the ith counterexample query.

This intuitively means that S_A halts within polynomial time w.r.t. the size of dfa and the length of counterexamples. Indeed, from (C.1) and (C.2), S_A can be regarded as a polynomial time "exact" query learning algorithm for DFA. (See the following section for details.)

4. Polynomial Time Learnability

In this section, we clarify the goal of learning problems and the notion of "polynomial time", thereby obtaining our formal definition for "polynomial time learnability".

We first define "exact learning", which is natural formalization for the learning achieved by many query learning algorithms [Ang87, BR87, Ish89, Sak88]. We point out that the goal of learning considered there may be too hard for many representation classes. Note that we have been able to prove that some of representation classes are polynomial time learnable in this strong learning notion; this is because we have been using (e.g., [Ang87]) (i) a weak definition for "polynomial time", and (ii) rather complicated query types. Here, by weakening the goal considered in "exact learning", we introduce two learning notions: "bounded learning" and "bounded statistical learning". Those learning notions are weaker than "exact learning", but they are still reasonable; furthermore, they provide more natural settings for studying polynomial time query learnability.

4.1. Exact Learning

The query learning algorithms that have been proposed in the literature are designed to achieve the following type of learning: For any learning system $\langle S, T \rangle$ and any representation class $C = (R, \Phi, \rho)$, we say that $\langle S, T \rangle$ *learns* C (or S *learns* C *from* T) if

(E.L) $\forall r \in R$, $\forall n \geq \rho(r)$
[$\langle S, T(r) \rangle(n) = r'$ such that $\Phi(r') = \Phi(r)$].

We refer to this type of learning as *exact learning*. Note that in the above learning condition we assume that n, which is called a *target size bound*, is given to S as an input.

We take this assumption since it is often assumed, and furthermore, it simplifies our discussions; however, such an assumption may not be essential for the exact learning.

In the exact learning, the notion of "polynomial time" is defined as follows: Let $C = (R, \Phi, \rho)$ be any representation class. For any functions t_1 and t_2, a learner S for C is called (t_1, t_2)-*time* if

$\forall T$: a teacher for C, $\forall r \in R$, $\forall n \geq \rho(r)$
[(i) $\#q_S(n; T(r)) \leq t_1(n)$, and
(ii) $\forall i, 0 \leq i \leq \#q_S(n; T(r))$ [$\#steps_S(n, i; T(r)) \leq t_2(\max(n, l_1, ..., l_i))$],
where l_j is the length of the answer to the jth counterexample query].

A learner is called *polynomial time* if it is (p, q)-time for some polynomials p and q. In a word, a polynomial time learner is a learning algorithm that halts within polynomially many steps in both a target size bound and the length of counterexamples.

Now we define "polynomial time learnability" in the exact learning. Let C be any representation class. For any teacher type α, we say that C is *polynomial time learnable from α-teachears* if

$\exists S$: a polynomial time learner,
$\forall T$: an α-teacher for C
[$\langle S, T \rangle$ learns C in the exact learning sense].

The above algorithm S is called a *polynomial time learning algorithm for C*.

For example, it follows from (C.1) and (C.2) at Example 3.1 and 3.2 that Angluin's algorithm is polynomial time learning algorithm for *DFA*; hence, *DFA* is polynomial time learnable from (Mem;Equ)-teachers.

Notice that when measuring learner's time complexity, we are considering not only an input, i.e., a bound for target representation size, but also the length of counterexamples. Because we include this latter factor when discussing "polynomial time", we have such a strange case that a polynomial time learner does not halt in polynomial time w.r.t. target representation size. On the other hand, this latter factor is in some sense *necessary*; without this, the polynomial time learnability notion (for the exact learning) does not make sense, or at least becomes very weak.

According to the learning condition (E.L), learning algorithms have to work with such unfriendly teachers that always reply with a very long counterexample; thus, it is almost obvious that we have to consider the length of counterexamples when measuring S's time

complexity because S cannot even read those counterexamples in "polynomial time" w.r.t. a target size bound. Even if we assume that teachers are friendly, i.e., they give shortest counterexamples, there is the case [Ang87] that the shortest counterexamples are much longer than target representations, and thus again, S cannot read those counterexamples in "polynomial time" w.r.t. a target size bound. We show one extreme case.

Proposition 4.1. There exists $\{r_i\}_{i\geq 1}$ such that

(i) $\{r_i\}_{i\geq 1} \subseteq R_{\mathrm{cfg}}$, and
(ii) $\forall f$: recursive, $\overset{\infty}{\forall} i$ [$|x_i| > f(|r_i|)$],
where x_i is one of the shortest elements in $\overline{\Phi_{\mathrm{cfg}}(r_i)}$.

Proof. Hartmanis constructed [Har79; Theorem 6] a class of context-free grammars $\{G_i\}_{i\geq 1}$ that generate regular sets $\{L_i\}_{i\geq 1}$ such that for every recursive function f and almost all i, we have $|M_i| > f(|G_i|)$, where M_i is the minimum state automaton for L_i. By an argument similar to his, we can prove that this $\{G_i\}_{i\geq 1}$ satisfies the proposition. □

This proposition shows that when learning $\{r_i\}_{i\geq 1}$, no recursive function can bound the length of the shortest nontrivial counterexample w.r.t. target representation size. Thus, if we consider a time bound that depends only on a target size bound, then any learning algorithm for *CFG* (that needs to read at least one nontrivial counterexample) has no recursive time bound. Note that this observation has two interpretations: (a) [Ang87] it is a justification for defining "polynomial time" by considering both target representation size and counterexample length, and (b) it illustrates that the goal considered in the exact learning is too hard for many representation classes. In the following discussions, we take the latter interpretation and search for weaker notions of learning.

4.2. Bounded Learning

One way to avoid the difficulty explained above and to use a more natural time bound, e.g., a time bound depending on a target size bound, is to give up obtaining a representation that denotes a target set *exactly*. Suppose that there is no counterexample of length $\leq 10^{1000}$ to what you conjectured; then you can be satisfied with your conjecture since you will find no trouble (in your life) by regarding it as an answer. This motivates the following type of learning: For any learning system $\langle S, T\rangle$ and any representation class $C = (R, \Phi, \rho)$, we say that $\langle S, T\rangle$ *learns* C if

(B.L) $\forall r \in R,\ \forall n \geq \rho(r),\ \forall m \geq 0$
$[\ \langle S, T(r)\rangle(\langle n, m\rangle) = r'$ such that $\Phi(r')^{\leq m} = \Phi(r)^{\leq m}\ ]$.

The second parameter m of an input is called a *length bound.* We refer to this type of learning as *bounded learning.*

The following point should be remarked here: In the bounded learning, although the learning condition does not insist on of a learner identifying a target set exactly, it still requires that a learner should obtain, for *any given* length bound m, a representation that denotes a target set up to length m. As shown in the following example, a length bound is sometimes regarded as the degree of precision.

Example 4.1. Consider the problem of learning representation class 1-$BALL = (R_{\text{1-ball}}, \Phi_{\text{1-ball}})$ defined at Example 2.2.

Intuitively, 1-$BALL$ represents 1-dimensional balls — closed intervals — in $[0,1]$; i.e., each $\langle x, d\rangle \in R_{\text{1-ball}}$ represents $[x-d, x+d]$. Recall that we consider only numbers that are encodable by the following simple coding scheme: for each $x \in [0,1]$, the code of x is the shortest $w \in \{0,1\}^*$ such that $0.w$ is a binary representation of x. Thus, more precisely, for each $\langle x, d\rangle \in R_{\text{1-ball}}$, the concept $\Phi_{\text{1-ball}}(\langle x, d\rangle)$ represented by $\langle x, d\rangle$ is the set $\{w \in \{0,1\}^* : w$ encodes a number in $[x-d, x+d]\}$, which is denoted by $\mathbf{B}[x-d, x+d]$. In the following we do not distinguish a string and the number it denotes; by "the digits of a number" we mean the length of the string denoting the number.

We show a learning algorithm S_B that learns 1-$BALL$ from (Mem;Sup)-teachers. The following is its outline.

prog S_B (**input** $\langle n, m\rangle$);
begin
{ We use X to denote a target set.}
(1) $v \leftarrow$ some point in X;
(2) $u \leftarrow$ some point in X such that $u - 1/2^m \notin X$;
(3) $w \leftarrow$ some point in X such that $w + 1/2^m \notin X$;
$d \leftarrow (w-u)/2$; $x \leftarrow u + d$;
output $\langle x, d\rangle$
end.

At (1), S_B obtains some point v in X by using one superset query. For example, S_B can ask whether $[0,0]$ $(= \Phi_{\text{1-ball}}(\langle 0,0\rangle))$ is a superset of X; if the answer is no, then it

can obtain some point in v as a counterexample (if yes, on the other hand, then S_B can conclude that X is $\mathbf{B}[0,0]$).

At (2), S_B computes u, a right extreme point of X with $1/2^m$ precision. Since some point v in X is already known, we can use the ordinary binary search to find u by using membership queries; for example, we first ask whether $u_1 = 0 \in X$?; if not, we ask whether $u_2 = (u_1 + v)/2 \in X$?; if so, we ask whether $u_3 = (u_2 + u_1)/2 \in X$; $\cdots$ until we obtain $u_k \in X$ such that $u_k - 1/2^m \notin X$. Furthermore, we can assume that no u' with less than m digits exists in $[u_k - 1/2^m, u_k] \cap X$. Similarly, S_B computes w, a left extreme point of X with $1/2^m$ precision.

Finally, S_B outputs $\langle x, d \rangle$ that represents the interval $\mathbf{B}[u,w]$. Note that the interval $\mathbf{B}[u,w]$ may not be exactly the same as X; however, it follows from the construction that every number in $X \triangle \mathbf{B}[u,w]$ has more than m digits. In other words, $\langle x, d \rangle$ represents X up to length m. Therefore, S_B learns 1-*BALL* in the bounded learning sense. Although an algorithm becomes more complicated, a similar idea works for any k-*BALL*, $k \geq 2$.

In the bounded learning, we measure the time complexity of a learner in terms of input n and m, i.e., target representation size and length bound. Let $C = (R, \Phi, \rho)$ be any representation class. For any function t from $\mathbf{N} \times \mathbf{N}$ to $\mathbf{N}$, a learner S for C is called *t-time* if

$$\forall T\text{: a teacher for } C,\ \forall r \in R,\ \ \forall n \geq \rho(r),\ \forall m \geq 0 \quad [\ time_S(\langle n, m \rangle; T(r)) \leq t(n,m)\ ].$$

A learner is called *polynomial time* if it is $\lambda nm.p(n+m)$-time for some polynomial p.

Since length of queries are not considered in the above definition, we have to assume that a teacher is "friendly" or "efficient", i.e., he always gives a reasonably short counterexample. Note that one can think of several different conditions for such "friendly" teachers. Here we present one such condition.

We first modify our learning system model so that a teacher also has access to an input tape. Then our condition is stated as follows: For any polynomial b (where $b(m) \geq m$ for all $m \geq 0$), a teacher T for C is called a *b-bounded teacher* if

$\forall r \in R,\ \forall n \geq \rho(r),\ \forall m \geq 0,$
$\forall q$: a counterexample query for which a counterexample of length $\leq b(m)$ exists
[$T(r)$ on $\langle n, m \rangle$ replies a counterexample of length $\leq b(m)$ to q]

Note that, for example, the set of b-bounded (Mem;Equ)-teachers for C is a proper subset of that of (Mem;Equ)-teachers for C. That is, the "b-bounded" notion is a restriction on teachers.

Now define "polynomial time learnability" in the bounded learning. Let C be any representation class. For any teacher type α, we say that C is *polynomial time learnable from polynomially bounded α-teachers* if

$\exists b$: a polynomial, $\exists S$: a polynomial time learner,
$\forall T$: a b-bounded α-teacher for C
[$\langle S, T \rangle$ learns C in the bounded learning sense].

The above algorithm S is called a *polynomial time learning algorithm for C*.

Example 4.2. The learning algorithm S_B explained at Example 4.1 gives an essential idea for learning 1-$BALL$ in polynomial time from polynomially bounded (Mem;Sup)-teachers.

Indeed, by modifying S_B, we can obtain a polynomial time learning algorithm that learns 1-$BALL$ from *every* (Mem;Sup)-teacher for 1-$BALL$. That is, we do not need the restriction "polynomially bounded" in this case; therefore, we claim that 1-$BALL$ is polynomial time learnable from (Mem;Sup)-teachers.

We have defined two polynomial time learnability notions: the one in the exact learning and the one in the bounded learning. Although one may expect, from the definition, that the former is stronger than the latter, such a relation is not always clear. The following is one of easily provable relations.

Proposition 4.2. For any representation class C, if C is polynomial time learnable from (Mem;Equ)-teachers in the exact learning sense, then it is polynomial time learnable from polynomially bounded (Mem;Equ)-teachers in the bounded learning sense. (The proof is easy; it is left to the interested reader.)

4.3. Bounded Statistical Learning

In our framework, we are assuming that teachers can *magically* return an appropriate answer. This assumption simplified learning problems and let us concentrate on studying learning strategies. On the other hand, we should be careful for not abusing this assumption. That is, we should be careful for not creating nor using complicated query types;

otherwise, we may be able to abuse the very powerful computation power of a teacher through those complicated queries. In this sense, we had better avoid using queries such as equivalence, subset, and superset queries[2]. However, it is provable [BW90] that one cannot achieve polynomial time learning with only membership queries for many representation classes. For example, the following negative result is provable as a corollary of [BW90].

Proposition 4.3. For every $k \geq 1$, *k-BALL* is not polynomial time learnable (in the bounded learning sense) from (Mem;)-teachers.

Remark Notice that no assumption such as P $\neq$ NP is necessary here. This is an absolute negative result.

Again we can draw two conclusions from such negative results: (a) those negative results justify using more complicated query types, and (b) they illustrate that the goal considered in the bounded learning is still too hard for many representation classes. Here we take the latter interpretation and discuss for a yet weaker learning notion.

One solution is to introduce the notion of "approximation". Note that we have already considered one type of approximation in the bounded learning. That is, the goal of the bounded learning is to obtain a representation r' that *approximates* a target set X in the sense that r' only need to denote X up to a given length bound. On the other hand, the set denoted by r' has to be *exactly* the same as X within the length bound. Here we relax this condition as follows: with *high probability* the set denoted by r' need to be *almost the same as* X within the length bound. In a word, the goal of our new learning notion is to obtain a "probably approximately correct" representation. Then we may be able to replace complicated queries with simple queries such as membership queries. (Indeed, Angluin [Ang87] has demonstrated this possibility for the problem of learning *DFA*.)

We prepare a framework for discussing "randomized computation" and "similarity of sets". For "randomized computation" we extend the notion of learning system so that learners can be randomized Turing machines. Although it is possible to consider

[2]Although it is hard to determine which query is "reasonable", we can consider that membership queries are of "reasonable" query types. As a matter of fact, we often use membership queries in natural sciences. For example, suppose that one physicist obtained some theory on the electrons, and he wanted to justify it; then he would do some experiments in order to find witnesses that support/disprove his theory. We can regard such experiments as "membership queries". On the other hand, it would be crazy if he asked 'Is my theory right?' to the electrons; we never expect that the electrons can answer "equivalence query".

"randomized teacher", we still assume that teachers are deterministic in our framework. For any event E concerning some randomized learning system, let $\Pr\{ E \}$ denote the probability that the event occurs.

In order to measure the similarity of two concepts, we have to introduce some measure or distribution on Σ^*. Thus, we extend the notion of representation class and define "distributed representation class". A *distributed representation class* is a six-tuple $(R, \Phi, \mu, \rho, \Gamma_1, \Gamma_2)$, where μ, called a *distribution functional* (or simply, a *distribution*), is a functional from $\mathbf{N}$ into $\Gamma_2^* \to [0,1]$, and the others are the same as ordinary representation classes. (Again we use Σ for Γ_1 and Γ_2, and omit specifying them.) In the following, for any $m \geq 0$, we use μ_m to denote the function $\mu(m)$; for any set A, we use $\mu_m(A)$ to denote $\sum_{w \in A} \mu_m(w)$.

Let $D = (R, \Phi, \mu, \rho)$ be any distributed representation class. Intuitively, for each $m \geq 0$, the function μ_m $(= \mu(m))$ is regarded as a distribution function over $\Sigma^{\leq m}$; that is, for every $w \in \Sigma^{\leq m}$, $\mu_m(w)$ is the probability that w appears as an instance of $\Sigma^{\leq m}$. Thus, μ must agree with the following condition.

$$\textbf{Condition for Distribution:} \quad \forall m \geq 0 \ [\ \mu_m(\Sigma^{\leq m}) = 1\ ].$$

Now our new learning notion is defined as follows: For any randomized learning system $\langle S, T \rangle$ and any distributed representation class $D = (R, \Phi, \mu, \rho)$, we say that $\langle S, T \rangle$ *learns* D if

$$\begin{aligned} \text{(S.L)} \quad & \forall r \in R,\ \forall n \geq \rho(r),\ \forall m \geq 0,\ \forall d, e > 0 \\ & [\ \langle S, T(r) \rangle(\langle n, m, d, e \rangle) = r' \\ & \quad \text{such that } \Pr\{\ \mu_m\{\ \Phi(r') \triangle \Phi(r)\ \} \geq 1/d\ \} \ <\ 1/e\ ]. \end{aligned}$$

The third and the forth input parameters are called an *approximation error bound* and a *computation error bound* respectively. We refer to this type of learning as *bounded statistical learning.*

The notion of "polynomial time learner" is defined almost in the same way as the bounded learning. Let $D = (R, \Phi, \mu, \rho)$ be any distributed representation class. For any function t from $\mathbf{N}^4$ to $\mathbf{N}$, a learner S for D is called t-time if

$$\forall r \in R,\ \forall n \geq \rho(r),\ \forall m \geq 0,\ \forall d, e > 0\ \ [\ time_S(\langle n, m \rangle; T(r)) \leq t(n, m, d, e)\ ].$$

A learner is called *polynomial time* if it is $\lambda nmde.p(n+m+d+e)$-time for some polynomial p. Finally, the notion of "polynomial time learnability" is defined in the same way as in the bounded learning.

Example 4.3. Consider the problem of learning 1-$BALL$ under some simple distribution.

One of the simplest distributions is the *uniform distribution* μ^{unif} that is defined as follows: for every $m \geq 0$ and $w \in \{0,1\}^{\leq m}$, $\mu_m^{\text{unif}}(w) = 1/(2^{m+1}-1)$. Let 1-$BALL$-$\mu^{\text{unif}}$ denote the representation class $\langle R_{\text{1-ball}}, \Phi_{\text{1-ball}}, \mu^{\text{unif}} \rangle$. We show that 1-$BALL$-$\mu^{\text{unif}}$ is polynomial time learnable from (Mem;)-teachers. (Note that this is not a uniform distribution on $\mathbf{B}[0,1]$ since $\mathbf{B}[0,1] \subsetneq \{0,1\}^*$; indeed, for sufficiently large m, we have $\mu_m^{\text{unif}}(\mathbf{B}[0,1]) \approx \frac{1}{2}$.)

The idea of polynomial time learner S_C for 1-$BALL$-μ^{unif} is easy. Recall the bounded learning algorithm S_B for 1-$BALL$; see Example 4.1. For learning a target interval X (on input $\langle n, m \rangle$), S_B uses only one superset query in order to obtain one point v in X. Here S_C randomly generates point v' in $[0,1]$ (more precisely, a string of length $\leq m$ in $\mathbf{B}[0,1]$) and checks whether v' is in X by a membership query. If S_C finds some $v' \in X$, then it can use v' as v and continue the computation of S_B; on the other hand, if it fails to find $v' \in X$ after sufficiently many experiments, which means that $\mu_m^{\text{unif}}(X)$ is small with high probability, then it can conclude that X is empty. (Technically, the empty set cannot be expressed by 1-$BALL$-μ^{unif}; but, S_C can output the representation of any small interval, e.g., $[0.,0.]$, as its answer.) Thereby, S_C can learn 1-$BALL$-μ^{unif} by using only membership queries. The detailed construction of S_C is left to the interest reader.

This example suggests that for representation classes with "simple" distributions, one can simulate complicated queries with simple queries such as membership queries. Indeed, such an intuition is verified in many cases [DW90].

Between two polynomial time learnability notions defined in the bounded learning and in the bounded statistical learning, we have the following relation, which is straight forward from the definition.

Proposition 4.4. For any teacher type α and any representation class $C = (R, \Phi, \rho)$, if C is polynomial time learnable from (polynomially bounded) α-teachers in the bounded learning sense, then for any distribution μ, $D = (R, \Phi, \mu, \rho)$ is polynomial time learnable from (polynomially bounded) α-teachers in the bounded statistical learning sense.

Notice that in our framework we can design learning algorithms *depending on* distributions; which is different from the present PAC learning model where learning algorithms should work for all the distributions. Of course, one can define a framework for "distribution independent statistical learning", but we think that the goal considered in such learning may be too difficult for many representation classes. Clearly, any learning algorithm that learns a representation class in the bounded learning sense can be regarded

as a distribution independent learning algorithm; for example, S_B at Example 4.1 works under any distribution because S_B makes no approximation error. We conjecture that this is only the way to design distribution independent learning algorithms unless we have a good method to detect a distribution; however, the problem of detecting a distribution from experiments is considered hard [AW90]. Thus, even if we introduce the distribution independent statistical learning, we may not be able to make learning goals easier than the ones in the bounded learning.

Acknowledgments

The preliminary version of this paper was presented at the workshop held by IIAS-SIS, Fujitsu Limited. The author thanks the participants of that workshop for their comments and suggestions. The preparation of this paper was done while the author was visiting Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya. The author would like to appreciate the kind support from the people at the department, and invaluable discussions with Prof. Balcázar, Prof. Díaz, and Prof. Gavaldà.

References

[AW90] N. Abe and M. Warmuth, On the computational complexity of approximating distributions by probabilistic automata, a manuscript. (*Need reconfirm.*)

[Ang80] D. Angluin, Finding patterns common to a set of strings, J. Comput. Syst. Sci. 21 (1980), 46-62.

[Ang87] D. Angluin, Learning regular sets from queries and counterexamples, Inform. and Comput. 75 (1987), 87-106.

[Ang88] D. Angluin, Queries and concept learning, Machine Learning 2 (1988), 319-342.

[AS83] D. Angluin and C. Smith, Inductive inference: theory and methods, Computing Surveys 15 (1983), 237-269.

[BR87] P. Berman and R. Roos, Learning one-counter languages in polynomial time, in "Proc. 28th IEEE Sympos. Foundations of Computer Science", IEEE (1987), 61-67.

[BW90] J. Balcázar and O. Watanabe, Some negative results on polynomial time query learnability, in preparation.

[BDG88] J. Balcázar, J. Díaz, and J. Gabarró, "Structural Complexity I", EATCS Monographs on Theoretical Computer Science, Springer-Verlag, Berlin (1988).

[BEHW87] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth, Classifying learn-

able geometric concepts with the Vapnik-Chervonenkis dimension, in "Proc. 18th ACM Sympos. on Theory of Computing", ACM (1986), 273-282; the final version will appear in J. Assoc. Comput. Mach.

[DW90] J. Díaz and O. Watanabe, Simulation of complicated queries by membership queries, manuscript.

[GS88] W. Gasarch and C. Smith, Learning via queries, in "Proc. 29th IEEE Sympos. Foundations of Computer Science", IEEE (1988).

[GMR85] S. Goldwasser, S. Micali, and C. Rackoff, The knowledge complexity of interactive proof systems, SIAM J. Comput. 18 (1989), 186-208.

[Har79] J. Hartmanis, On the succinctness of different representations of languages, in "Proc. 6th International Colloquium on Automata, Languages and Programming", Lecture Notes in Computer Science 71, (1979), 282-288.

[HU79] J. Hopcroft and J. Ullman, "Introduction to Automata Theory, Languages, and Computation", Addison-Wesley, Reading (1979).

[Ish89] H. Ishizaka, Learning simple deterministic languages, in "Proc. of 2nd Workshop on Computational Learning Theory", Morgan Kaufmann (1989), 162-174.

[PV88] L. Pitt and L. Valiant, Computational limitations on learning from examples, J. Assoc. Comput. Mach. 35 (1988), 965-984.

[PW88] L. Pitt and M. Warmuth, Reductions among prediction problems: on the difficulty of prediction automata, in "Proc. 3rd Structure in Complexity Theory Conference", IEEE (1988); the final version will appear in J.C.S.S.

[Sak88] Y. Sakakibara, Learning context-free grammars from structural data in polynomial time, in "Proc. 1st Workshop on Computational Learning Theory", Morgan Kaufmann (1988), 330-344.

[Tze89] W. Tzeng, The equivalence and learning of probabilistic automata, in "Proc. 30th IEEE Sympos. Foundation of Computer Science", IEEE (1989), 268-273.

[Val84] L. Valiant, A theory of the learnable, Commun. Assoc. Comput. Mach. 27 (1984), 1134-1142.

[War89] M. Warmuth, Towards representation independence in PAC learning, Lecture Notes in AI 397, Springer-Verlag (1989), 78-103.

[Wat90] O. Watanabe, A formal study of learning via queries, in "Proc. 17th International Colloquium on Automata, Languages and Programming", Lecture Notes in Computer Science 443, Springer-Verlag, Berlin (1990), 137-152.