Reductions Among Prediction Problems: On the Difficulty of Predicting Automata

Leonard Pitt*
University of Illinois
at Urbana-Champaign

Manfred K. Warmuth**
University of California
at Santa Cruz

Abstract

Consider the following question: Given examples of words accepted and rejected by an unknown automaton, is there an algorithm that in a feasible amount of time will learn to predict which words will be accepted by the automaton? We develop a notion of prediction-preserving reducibility, and show that if DFAs are predictable, then so are all languages in logspace. In particular, the predictability of DFAs implies the predictability of all boolean formulas. Similarly, predictability of NFAs and PDAs (or CFGs) implies predictability of all languages in nondeterministic logspace and $\mathcal{LOGCFL}$, respectively. We give relationships between the complexity of the membership problem for a class of automata, and the complexity of the prediction problem. Examples are given of prediction problems whose predictability would imply the predictability of all languages in $\mathcal{P}$. Assuming the existence of 1-way functions, it follows from [13] that these problems are not predictable even in an extremely weak sense.

1 Introduction

Concepts are languages over a finite alphabet and *concept classes* are families of languages. *Examples* of a concept are words tagged with a label saying whether or not the word is in the concept. A class of concepts is *predictable* iff there is a polynomial time algorithm A such that for any concept in the class and for any fixed unknown probability distribution on words, if a polynomial number of examples generated according to the distribution is given to A, then with high probability, A predicts correctly on a new word drawn from the fixed distribution.

Following the research of Valiant [29] on distribution independent learning, we require that A must perform well on any concept in the class (worst case), and on any distribution (distribution independent convergence). This notion of predictability (introduced in [19]) is a generalization of Valiant's definition of learnability [29], and the learnability of a concept class implies the predictability of the class [18].

The results of this paper concentrate on concept classes for which no feasible prediction (or learning) algorithms are known, such as the concept classes defined by boolean formulas, DFAs,[1] NFAs, PDAs, CFGs, and resource bounded Turing machines.

We give formal definitions for prediction problems and introduce a notion of polynomial time prediction-preserving reducibility among prediction problems. Intuitively, a polynomial time prediction-preserving reduction consists of two mappings: a polynomial time computable function f that maps unlabelled examples of the first problem to unlabelled examples of the second problem, and a function g that maps concepts of the first problem to concepts of the second problem. An interesting feature of our definition of reduction is that the mapping g need not be computable. We only require that g be length preserving within a polynomial.

Our definition of reduction is very similar to the ones given in [24,25], except that we allow more general domains and we add requirements that ensure that the reduction preserves polynomial time predictability.

Using our notion of reduction, we show that the predictability of DFAs implies the predictability of all concepts accepted by deterministic logspace Turing ma-

*Supported by the Department of Computer Science, University of Illinois at Urbana-Champaign. Part of this work was done while the first author was visiting the Department of Computer and Information Sciences, University of California at Santa Cruz. Author's address: Dept. of Computer Science, University of Illinois, Urbana, IL 61801.

**Supported by ONR grant N00014-86-K-0454. Author's address: Dept. of Computer and Information Sciences, University of California, Santa Cruz, CA 95064.

[1]Angluin gives a learning algorithm for DFAs that relies on the ability to make queries as to the membership of examples chosen by the algorithm [2].

chines. A corollary of this result is that any algorithm for predicting DFAs can be used to predict any boolean formula.

We also show in a similar manner that the predictability of NFAs implies the predictability of any concept accepted by a nondeterministic logspace Turing machine, and that predictability of CFGs in Chomsky normal form implies the predictability of any concept (language) in $\mathcal{LOGCFL}$.

We give natural prediction problems that, if predictable, would imply the predictability of all concepts accepted in polynomial time. It follows from the work of Goldreich, Goldwasser, and Micali [13] that these prediction problems are not predictable (even in an extremely weak sense) assuming the existence of 1-way functions. Our list of such hard prediction problems includes Convex Polytope Intersection (the concept is induced by a hidden convex polytope, positive examples are subcubes that have non-empty intersection with the polytope), and Horn Clause Consistency (the concept is induced by a hidden conjunction of Horn clauses, positive examples are sets of facts that are consistent with the hidden conjunction).

Our results highlight intriguing connections between polynomial predictability, cryptography, and complexity theory.

2 Background

A major tool for studying distribution free convergence is the Vapnik-Chervonenkis dimension [20] (following [30]) of the concept class. Consider the following three cases: (1) the VC dimension is fixed for all concepts in the class; (2) the VC dimension grows polynomially in some natural parameters of the concepts in the class (e.g., in the size of some encoding of concepts in the class); (3) the VC dimension grows more than polynomially in the parameters of the concepts in the class.

When (1) is the case, and the class is polynomially recognizable,[2] then there is a canonical polynomial time prediction algorithm [19] based on the "1-inclusion graph". When (3) is the case, it has been shown that no polynomial time prediction algorithm is possible [6,7,11]

More relevant to our investigation is the case (2). In this situation a number of methods have led to feasible prediction algorithms: (*i*) Find a polynomial time algorithm that, when given any set of examples as input, produces a hypothesis consistent with the examples and for which the VC dimension of the hypothesis class is polynomial in the VC dimension of the original class. In this case, the concept class is polynomially learnable [6,7], as well as polynomially predictable [19]. (*ii*) Find a polynomial time prediction algorithm whose worst case total number of prediction mistakes in any sequence of predictions grows polynomially in the parameters of the concepts [25]. (*iii*) Find a prediction algorithm whose permutation index [19] grows polynomially with the parameters of the concepts. The learnability and predictability of the concept classes that we investigate are still open questions, thus none of the above approaches has yet been successfully applied to these problems.

[2]A concept class is polynomially recognizable iff there is a polynomial time algorithm that when given any set of examples as input determines whether there is some concept in the class that is consistent with the examples.

As for *nonlearnability* results for DFAs, this problem has been studied (using various different models of learning) in [3,12], among others. Gold [12] shows that it is NP-hard to find the smallest DFA consistent with a given sample. Angluin [3] shows that this is true even if all but ϵ of the words up to a given length are given as examples. Thus if "learnable" requires that a minimum size automaton equivalent to the unknown automaton be found, then there are nonlearnability results. Nonetheless, it is still conceivable that an algorithm exists for learning DFAs in the sense of Valiant [29], since there is no requirement that the DFA found be as small as the DFA to be learned. (In fact, it may be polynomially larger, and even accept a slightly different language.)

For nonlearnability within Valiant's paradigm, Pitt and Valiant [27] have shown that (unless R = NP), for each $k \geq 2$ there is no learning algorithm that, if given as input randomly generated examples of any k-term DNF concept, will be able to find an approximate k-term DNF hypothesis. Actually, it is shown that this type of nonlearnability holds even if the learning algorithm is allowed to ouput a hypothesis with roughly $2k$ terms. However, these results rely on the syntactic constraints imposed by the requirement that the formula output by the learning algorithm (the hypotheses of the algorithm) must be expressed in some particular representation. If these constraints are relaxed, there *is* a learning algorithm for k-term DNF that outputs hypotheses in k-CNF that approximate the k-term DNF formula to be learned [27]. Haussler [15] gives similar nonlearnability results that rely on representational constraints.

The notion of predictability introduced by Haussler, Littlestone, and Warmuth [19], and studied here, captures learnability when the learning algorithm is allowed to ouput a hypothesis in *any* representation. Consequently, our hardness results are more meaningful: We show, based on cryptographic assumptions, that certain concept classes are not polynomially predictable, and thus are certainly not polynomially learnable. For other problems such as DFA prediction, we have not been able to prove unpredictability even with cryptographic assumptions. However, we provide evidence of the difficulty of prediction for these problems by showing they are "prediction-complete" for a large class of prediction problems.

A similar notion of reduction between prediction problems in boolean domains was used by Littlestone [25] for a different model of predictability discussed in Section 6. Reductions were given from various boolean function classes to the classes of monotone monomials and linearly separable boolean functions, for which efficient algorithms with small worst case mistake bounds were developed. Kearns et al [24] give reductions between various *learning* problems in the boolean domain, and Haussler [15] gives reductions between learning problems in structural domains.

3 Definitions

Throughout the paper, let Σ and Γ be fixed, finite alphabets. $\mathcal{LOG}$, $\mathcal{NLOG}$, and $\mathcal{P}$ denote the classes of languages accepted in deterministic logspace, nondeterministic logspace, and polynomial time, respectively.

Definition 1 *A* concept *c is any subset of* Σ^*. *A* concept class *C is any collection of concepts.*

We are interested in characterizing those concept classes that are "polynomially predictable", i.e., concept classes C for which there exists a polynomial time algorithm that, given polynomially many randomly generated elements of Σ^*, and told for each word whether or not the word is in some unknown concept $c \in C$, can predict with high probability whether a new (unseen) word is in c.

There are many issues involved in defining the predictability of a concept class. A main one is that it is desirable to allow a prediction algorithm to receive more training examples (and to spend more time) before achieving accurate prediction, depending on the "complexity" of the concept to be predicted. A reasonable measure of this complexity is the length (number of letters) of the description of the concept in some given representation. Thus the predictability will depend on what type of representation of the concept we have chosen. For example, we may choose to represent regular languages by DFAs, NFAs, regular expressions, etc. We would like to ask the question "Are DFAs predictable" rather than the question "Are regular languages predictable". This motivates the following definition.

Definition 2 *A* prediction problem *is a pair* (R, c), *where* $R \subseteq \Gamma^*$, *and c is a mapping* $c : R \rightarrow 2^{\Sigma^*}$. *R is a called a "set of representations", and each* $r \in R$ *denotes the concept* $c(r) \subseteq \Sigma^*$. *The concept class represented by* (R, c) *is* $C(R, c) = \{c(r) : r \in R\}$.

Generally, given any R, the mapping c will be implicitly understood, hence we use R as an abbreviation for (R, c). We will also refer to R alone as a prediction problem.

In the following definitions of particular prediction problems, we use the word "encodes" to abbreviate "encodes with respect to some fixed, standard encoding scheme." As usual, if numbers are used to define the concepts, then we must explicitly mention whether they are to be encoded in unary or binary.

- $R_{DFA} = \{r \in \Gamma^* : r$ encodes a DFA$\}$ is a set of representations (for the concept class of regular languages) with the implicit mapping c such that for any r, $c(r)$ is the concept (language) accepted by the DFA encoded by r.
- $R_{NFA} = \{r : r$ encodes an NFA$\}$.
- $R_{PDA} = \{r : r$ encodes a PDA$\}$.
- $R_{CFG} = \{r : r$ encodes a CFG in Chomsky normal form$\}$.
- $R_{BF} = \{r : r$ encodes a boolean formula$\}$. In this case, given a formula r of n variables, the concept $c(r)$ is exactly those words of length n that, when interpreted as an assignment to the n variables, satisfy the formula encoded by r.
- $R_{CIRC} = \{r : r$ encodes a boolean circuit$\}$, where if r has n inputs, then $c(r)$ is the set of boolean input strings of length n accepted by the circuit encoded by r.

In the definition below, we allow a prediction algorithm time polynomial in the input parameter s, which is an upper bound on the length of the representation r of the unknown concept $c(r)$. Without sacrificing polynomial predictability, this parameter need not be explicitly given to the algorithm. (Details omitted.) It is included for clarity of argument.

Another issue is that the languages accepted by DFAs, NFAs, etc., contain words of different lengths, and the minimum number of examples sufficient for accurate prediction should be allowed to grow with the length of the input examples. For this reason, if the prediction algorithm does not receive any information about the lengths of words likely to occur according to the unknown distribution, then there are subtle issues involved in defining polynomial predictability in a natural way. A more detailed discussion is omitted from this abstract. Here we will assume that for some m, the probability of example words of length greater than m is zero. The prediction algorithm is given m as a parameter and will be allowed a number of examples polynomial in m to achieve accurate prediction. (There are other ways to deal with this issue: For example, the prediction algorithm could be told a length m such that the probability of a word of length more than m occuring is at most γ (for some small γ.))

Definition 3 *For any language L, let $L^{[m]} = \{w \in L : |w| \leq m\}$.*

Definition 4 *For any concept c and word w, let $label_c(w)$ = "+" if $w \in c$ and "−" if $w \notin c$. An* example *of c is a pair $\langle w, label_c(w)\rangle$.*

Definition 5 *A* prediction algorithm *A is an algorithm that takes as input three parameters s, m, and ϵ, a collection of elements of $\Sigma^{[m]} \times \{+,-\}$, and an element $w \in \Sigma^{[m]}$. The output of A is either "+" or "−", indicating its prediction for w. A is a* polynomial time prediction algorithm *if there exists a polynomial t such that the run time of A is at most $t(s, m, \frac{1}{\epsilon}, l)$, where l is the total length of the input of A.*

Definition 6 *The prediction problem R is* (polynomially) predictable *iff there exists a polynomial time prediction algorithm A and polynomial p such that for all input parameters s, m, and $\epsilon > 0$, for all $r \in R^{[s]}$, and for all probability distributions on $\Sigma^{[m]}$, if A is given $p(s, m, \frac{1}{\epsilon})$ randomly generated examples of (the hidden concept) $c(r)$, and a randomly generated unlabelled word $w \in \Sigma^{[m]}$, then the probability that A incorrectly predicts $label_{c(r)}(w)$ is at most ϵ.*

Comments: From here on, "predictable" means "polynomially predictable". Our model is based on the definition of predictability introduced in [19]. (In this paper we also allow the number of examples to grow with m for the reasons discussed above.) These definitions of predictability are analogous to the learnability model introduced by Valiant [29]. The correspondence is as follows: Let *error(A)* denote the probability that A predicts incorrectly on the unlabelled word. Predictability requires that $error(A) \leq \epsilon$. In the learnability model, the algorithm must output a hypothesis $r' \in R$ such that the probability (with respect to the fixed distribution) of the symmetric difference of $c(r)$ and $c(r')$ (i.e., the "error" of $c(r')$) is at most ϵ with probability at least $1-\delta$ (for arbitrarily small parameters ϵ, δ). Polynomial predictability is equivalent to the existence of a polynomial time learning algorithm that is allowed to output *any polynomial time algorithm* as a representation for the hypothesis [18].

We now define prediction-preserving reductions. Intuitively, R reduces to R' iff for every representation r in R there is an at most polynomially larger representation $g(r)$ in R' such that a word w is in $c(r)$ iff a polynomially computed transformed word $f(w)$ is in $c(g(r))$.

Definition 7 *Let R and R' be prediction problems. Then R reduces to R' (denoted by $R \trianglelefteq R'$) iff there are functions $f : \Sigma^* \times \mathbb{N} \times \mathbb{N} \rightarrow \Sigma^*$ (the word transformation) and $g : R \times \mathbb{N} \rightarrow R'$ (the representation transformation), and polynomials t and q such that for all $s, m \in \mathbb{N}$, $r \in R^{[s]}$, and $w \in \Sigma^{[m]}$,*

(1) *$w \in c(r)$ iff $f(w, s, m) \in c(g(r, m))$;*

(2) *f is computable in time $t(|w|, s, m)$ (and thus $|f(w, s, m)| \leq t(|w|, s, m)$);*

(3) *$|g(r, m)| \leq q(|r|, m)$.*

Note that the representation transformation g need not be polynomial-time computable. In fact, it need not be computable. We only require that it is length preserving within a polynomial. We discuss relaxations of this definition in Section 6.

4 Automata Prediction

Theorem 8 *For all prediction problems R and R', if $R \trianglelefteq R'$ and R' is predictable, then R is predictable.*

Proof: Let A' be a polynomial time prediction algorithm for R', and let $R \trianglelefteq R'$, with word transformation f, representation transformation g, and polynomials t and q as above. Without loss of generality, we assume that t and q are monotone nondecreasing in each parameter. We construct a polynomial time prediction algorithm A for R. A simply takes its input parameters s, m, and ϵ, and passes the parameters $q(s, m), t(m, s, m)$, and ϵ to A'. Then for each example

$(x, label)$, A computes $f(x,s,m)$ and passes the example $(f(x,s,m), label)$ to A'. Note that the examples A gives to A' are examples of some concept whose representation $g(r,m) \in R'$ has length at most $q(s,m)$. A also gives the unlabelled word $f(w,s,m)$ to A', where w is the unlabelled word that A receives. To predict the label of w, A predicts exactly what A' predicts for the word $f(w,s,m)$.

A' predicts incorrectly with probability at most ϵ (with respect to the image under f of the original distribution) when given $p(s, m, \frac{1}{\epsilon})$ examples. By the definition of a prediction preserving reduction, the probability that A predicts incorrectly is also at most ϵ when given $p(q(s,m), t(m,s,m), \frac{1}{\epsilon})$ examples. This number of examples and the running time of A is polynomial in s, m, and $\frac{1}{\epsilon}$. □

Definition 9 *If R is a prediction problem, and $\mathcal{R}$ is a set of prediction problems, then R is* prediction-hard *for $\mathcal{R}$ iff for all prediction problems $R' \in \mathcal{R}$, $R' \trianglelefteq R$. If $R \in \mathcal{R}$ also, then R is* prediction-complete *for $\mathcal{R}$.*

Associated with any prediction problem R is an *evaluation problem*, which is that of determining, given an arbitrary $r \in R$ and $w \in \Sigma^*$, whether or not $w \in c(r)$. This is defined formally as a language:

Definition 10 *The* evaluation problem *for a prediction problem R is the language $E(R) = \{(r,w) : w \in c(r)\}$.*

It will be useful to classify prediction problems based on the complexity of their evaluation problems. (Intuitively, there should be a relationship between the difficulty of determining membership in a concept when given a representation of the concept (the evaluation problem), and the problem of predicting membership in an unknown concept when using the same representations for the concepts (the prediction problem.))

Definition 11 *For a complexity class $\mathcal{L}$, let $\mathcal{R}_{\mathcal{L}} = \{R : E(R) \in \mathcal{L}\}$.*

We will show that R_{DFA} (R_{NFA}) is prediction-complete for $\mathcal{R}_{\mathcal{LOG}}$ ($\mathcal{R}_{\mathcal{NLOG}}$). Thus the predictability of DFAs (NFAs) implies the predictability of any prediction problem whose evaluation problem is in $\mathcal{LOG}$ ($\mathcal{NLOG}$).

Theorem 12 *R_{DFA} is prediction-complete for $\mathcal{R}_{\mathcal{LOG}}$.*

Theorem 12 is proved by showing that predictability of DFAs implies the predictability of logspace Turing machines. We need the following definition.

Definition 13 *Let $\Delta = \Sigma \cup \Gamma \cup \{B\}$ (where B denotes the blank symbol). For each constant $k > 0$, let $R_{k \log TM} = \{r : r$ encodes a single tape (offline) TM with tape alphabet Δ that runs in time at most $k \log n$ on inputs of length $n\}$.*

Lemma 14 *Let $R \in \mathcal{R}_{\mathcal{LOG}}$. Then for some $k > 0$, $R \trianglelefteq R_{k \log TM}$.*

Proof: Let $R \in \mathcal{R}_{\mathcal{LOG}}$. Then for some constant k, there is a single tape, $k \log n$ space bounded TM T with tape alphabet Δ that accepts $E(R)$. For each $r \in R$ there is a single tape, $k \log n$ space bounded TM T_r with tape alphabet Δ, and of size $\mathcal{O}(|T| + |r|)$, that accepts $c(r)$. Hence f and g witness that $R \trianglelefteq R_{k \log TM}$, where for any s, r, m, and w, $f(w,s,m) = w$ and $g(r,m) = T_r$. □

Proof of Theorem 12: It is easy to show that $R_{DFA} \in \mathcal{R}_{\mathcal{LOG}}$. To show that any $R \in \mathcal{R}_{\mathcal{LOG}}$ reduces to R_{DFA} we show that for each fixed $k > 0$, $R_{k \log TM} \trianglelefteq R_{DFA}$, and the result follows from Lemma 14 and transitivity of $\trianglelefteq$.

To show that $R_{k \log TM} \trianglelefteq R_{DFA}$ it suffices to show that there exist polynomials t and q such that for every TM $T \in R_{k \log TM}$ (with state set Q_T) and for each $m > 0$, there exists a DFA M of size at most $q(|Q_T|, m)$ and a polynomial p such that for any $w \in \Sigma^{[m]}$, M accepts a transformed word $f(w, |Q_T|, m) = 1^{|w|}0w^{p(|w|,|Q_T|,m)}$ iff T accepts w.

Note that f simply replicates the word w some (polynomial) number of times, and precedes it with a string of 1's of length $|w|$ followed by a 0. The DFA M will, on input $f(w, |Q_T|, m)$, simulate the action of T on input w.

M is composed of $m + 1$ smaller DFAs, M_0, M_1, ..., M_m. M uses a chain of $m + 1$ states to read the initial string of 1's in $f(w, |Q_T|, m)$, and branch to $M_{|w|}$ upon seeing the first 0. $M_{|w|}$ is a machine designed to simulate T on inputs of length exactly $|w|$. The behavior of $M_{|w|}$ on the $p(|w|, |Q_T|, m)$ copies of w will simulate the behavior of T on w.

In essence, the DFA $M_{|w|}$ must overcome two obstacles: that the Turing machine T uses work space $k \log |w|$, and that it can move in two directions. $M_{|w|}$ overcomes the first obstacle by having polynomially many states to encode the memory of T. The second obstacle is overcome by using the repeated copies of w to avoid moving left.[3]

[3]In a similar fashion in [28] a power of ww^R is used (where w^R is the reverse of w) to reduce two-way head movement to

$M_{|w|}$ stores the entire contents of the worktape of T into its state information and simulates T. The only problem occurs when T tries to move to the left. In that case, $M_{|w|}$ simply moves to the corresponding symbol in the next copy of the input word by moving $|w| - 1$ symbols to the right. Thus $M_{|w|}$ has as part of its state information a mod $|w| - 1$ counter that it invokes whenever T moves left. The number of states of $M_{|w|}$ is at most the product of the number of states of T, the number of states needed for the counter, the number of distinct worktape configurations of T, and the number of possible worktape head positions. This product is at most

$$(|Q_T|)(|w| - 1)(|\Delta|^{k \log |w|})(k \log |w|)$$

and the number of states of M is at most

$$m + 1 + (m+1)(|Q_T|)(m-1)(|\Delta|^{k \log m})(k \log m).$$

Thus the length of the encoding of M is at most polynomial in m and the length of the encoding of T. Further, the number $p(|w|, |Q_T|, m)$ of copies of w that are required is at most the number of moves of T on an input of length m, which is at most a polynomial in m and $|Q_T|$, since T is space bounded by $k \log m$. □

The following corollary shows that DFA predictability implies the predictability of boolean formulas.

Corollary 15 $R_{BF} \trianglelefteq R_{DFA}$.

Proof: Since R_{DFA} is complete for $\mathcal{R}_{\mathcal{LOG}}$, we need only show that $E(R_{BF})$ is in $\mathcal{LOG}$, i.e., that given a boolean formula of n variables followed by a bit string w, it is possible to determine in logarithmic space whether w has length n and corresponds to a satisfying assignment for the formula. We can easily determine whether the string has the correct length in logspace. Lynch [26] shows that given a variable free boolean formula, it is possible to evaluate it in logspace. (Buss [8] strengthens this to alternating logtime). We must show that if the inputs are not hardwired into the formula, but given after the formula as a bit string, then it is still possible to evaluate the formula in logspace. This is easily achieved using a number of counters to index into the assignment string and fetch the corresponding assignment to a symbolic variable that is encountered while running Lynch's algorithm on the boolean formula containing variables. □

The proof of Theorem 12 actually works for any reasonable definition of "automaton". Rather than formalizing the notion of "a class of automata", for the

one-way head movement.

purposes of this presentation, we let "a class of automata" be any of $\{DFA, NFA, PDA\}$.

Theorem 16 *If A is a class of automata, then let $p2A{+}k\log$ be the class of languages accepted by polynomial time bounded 2-way versions of automata of type A with additional work space $k \log n$. Let $\mathcal{R}_{A,k} = \{R : E(R) \in p2A + k\log\}$. Then R_A is prediction-hard for $\cup_k \mathcal{R}_{A,k}$.*

By letting $A = DFA, NFA, PDA$, Theorem 16 states that R_{DFA} is prediction-hard for $\mathcal{R}_{\mathcal{LOG}}$, that R_{NFA} is prediction-hard for $\mathcal{R}_{\mathcal{NLOG}}$, and that R_{PDA} is prediction-hard for $\mathcal{R}_{\mathcal{LOGCFL}}$ (where $\mathcal{LOGCFL}$ is the class of languages accepted by polynomial time bounded 2-way PDAs with auxiliary logspace work tape). The proof is the same as that of Theorem 12.

Corollary 17 *R_{NFA} is prediction-complete for $\mathcal{R}_{\mathcal{NLOG}}$.*

For each polynomial q, let
$R_{qPDA} = \{r : r$ encodes a $q(n)$ time bounded $PDA\}$.

Corollary 18 *R_{CFG} is prediction-complete for $\mathcal{R}_{\mathcal{LOGCFL}}$, and there exists a polynomial q such that R_{qPDA} is prediction-complete for $\mathcal{R}_{\mathcal{LOGCFL}}$.*

Proof: R_{PDA} is prediction-hard for $\mathcal{R}_{\mathcal{LOGCFL}}$ by Theorem 16. For any PDA M there is a CFG G in Chomsky normal form[4] of size polynomial in the size of M, that generates the same language. It follows that R_{CFG} is also prediction-hard for $\mathcal{R}_{\mathcal{LOGCFL}}$. (The representation transformation maps a PDA to the equivalent CFG, and the word transformation is the identity.) Prediction-completeness for R_{CFG} follows by showing that its evaluation problem is in $\mathcal{LOGCFL}$, proving the first part of the corollary. Note that to derive a word of length $n > 0$ in Chomsky normal form takes exactly $2n-1$ steps (1 step if $n = 0$). Therefore, there are polynomials p and q such that for each grammar G there is a PDA M_G of size at most $p(|G|)$ such that $\{w : M_G$ accepts w in at most $q(|w|)$ steps$\} = L(G)$. It follows that $R_{CFG} \trianglelefteq R_{qPDA}$, and thus R_{qPDA} is prediction-hard for $\mathcal{R}_{\mathcal{LOGCFL}}$. Finally, there is a single "simulation" polynomial time 2-way auxiliary PDA using logspace worktape that accepts the language $E(R_{qPDA})$. Thus R_{qPDA} is prediction-complete for $\mathcal{R}_{\mathcal{LOGCFL}}$. □

We conclude this section by discussing some differences between our notion of prediction-preserving

[4]We use the definition in [14], which allows the production $S \rightarrow \lambda$.

reduction, and the standard many-one deterministic logspace reduction used in complexity theory. We showed that the prediction problem R_{DFA} is prediction-complete for $\mathcal{R}_{\mathcal{LOG}}$. Following [22], $E(R_{DFA})$ (the membership problem for DFAs) is complete for $\mathcal{LOG}$ with respect to one-way, read-only-input, deterministic logspace reductions. The analogous facts hold for R_{NFA}, $\mathcal{R}_{\mathcal{NLOG}}$, $E(R_{NFA})$ and $\mathcal{NLOG}$, but with respect to standard deterministic logspace reductions. Given these observations, a tempting conjecture might be that if an evaluation problem $E(R)$ is complete (with respect to logspace reductions) for some complexity class $\mathcal{L}$, then R is prediction-complete for $\mathcal{R}_{\mathcal{L}}$.

This conjecture is not true because prediction-preserving reductions are sufficiently different from the standard reductions between languages. If $E(R)$ is complete for $\mathcal{L}$ then for any $L \in \mathcal{L}$ there is *one* logspace computable function f such that for all u, $u \in L$ iff $f(u) \in E(R)$. In our notion of reduction, *two* functions are required, one for the word transformation and one for the representation transformation. For any language L that is complete (with respect to logspace reductions) for some complexity class $\mathcal{L}$, the language $L \times \{1\}$ is also complete for $\mathcal{L}$. It is easy to define a set of representations R such that $E(R) = L \times \{1\}$: Let $R = \Sigma^*$, where any word r represents the concept $\{1\}$ if $r \in L$, otherwise, r represents the concept $\{\}$. Now $E(R)$ is complete for $\mathcal{L}$, but R is trivially predictable.

5 Unpredictable Concepts

In this section we give a number of problems that are prediction-complete for $\mathcal{R}_{\mathcal{P}}$ and discuss why such problems are not expected to be predictable.

Theorem 19 *R_{CIRC} is prediction-complete for $\mathcal{R}_{\mathcal{P}}$.*

Proof: Let R be any prediction problem such that $E(R) \in \mathcal{P}$. To show that $R \trianglelefteq R_{CIRC}$, note that for each representation $r \in R$ and each length bound m there is a boolean circuit B_r that accepts all words of $c(r)$ of length at most m, padded so as to have equal length. More precisely, let B_r accept the language $\{w01^{m-|w|} : |w| \leq m \text{ and } w \in c(r)\}$. Thus B_r has $m+1$ inputs. Since $E(R) \in \mathcal{P}$, there exists a polynomial q such that for all $r \in R$ and $m \in \mathbb{N}$, B_r can be chosen such that $|B_r| \leq q(|r|, m)$. □

Goldreich, Goldwasser, and Micali [13] have shown, assuming the existence of any 1-way function, how to construct polynomial time computable functions that are hard to predict with respect to a very weak notion of prediction. They describe a polynomial time TM T of two inputs of the same length. Call the first input the index. Each index $r \in \Sigma^m$ defines a new TM T_r such that $T_r(w)$ accepts iff $T(r, w)$ accepts. Note that $|T_r|$ is $O(|r|)$. Let $R_{HARD(m)}$ = the set of encodings of machines T_r for each $r \in \Sigma^m$. Let $R_{HARD} = \cup_m R_{HARD(m)}$. Since each T_r is essentially the same polynomial time Turing machine T, $E(R_{HARD}) \in \mathcal{P}$.

Goldeich et al prove that R_{HARD} has the following property. Let A be any polynomial time algorithm (possibly randomized) that works as follows: A, given m as input, attempts to predict those words of length m accepted by some randomly chosen element $r \in R_{HARD(m)}$ by first *querying* any collection of words of length m as to whether or not the machine encoded by r accepts. Then A chooses a different test word w (of length m) and predicts whether w is accepted by the machine encoded by r. Let *avg-error$_m$*(A) denote the probability that A is incorrect, where the probability is taken over the random selection of $r \in R_{HARD(m)}$ according to a uniform distribution (and over all possible runs of A, if A is randomized). Then for any such algorithm A, and any polynomial p, *avg-error$_m$*$(A) \geq \frac{1}{2} - \frac{1}{p(m)}$, for sufficiently large m.

The consequences of this remarkable theorem is that the prediction problem R_{HARD} is not predictable. In particular, for any polynomial prediction algorithm A for R_{HARD}, for any polynomial p, and for sufficiently large m, there exists a particular $r \in R_{HARD(m)}$ such that for w uniformly chosen from Σ^m, the probability that A incorrectly predicts $label_{c(r)}(w)$ is at least $\frac{1}{2} - \frac{1}{p(m)}$.

Thus R_{HARD} (which is an element of $\mathcal{R}_{\mathcal{P}}$) is certainly not predictable using our definition of predictability and the related ones given in [18]. Similarly, it has been shown assuming the existence of 1-way functions, that polynomial size boolean circuits are not polynomially learnable in the Valiant model [5]. The above discussion is summarized in the following theorem.

Theorem 20 *If there exists a 1-way function, then any prediction problem that is prediction-hard for $\mathcal{R}_{\mathcal{P}}$ is not predictable.*

Our goal now is to reduce R_{CIRC} to other natural prediction problems, thus showing that they are prediction-complete for $\mathcal{R}_{\mathcal{P}}$, and hence not predictable by Theorem 20. As was pointed out at the end of the previous section, if a language is $\mathcal{P}$-complete (with respect to logspace reductions) then a given related prediction problem is not necessarily prediction-complete for $\mathcal{R}_{\mathcal{P}}$. Nonetheless, a number of $\mathcal{P}$-complete evaluation problems *do* have related prediction problems that are prediction-complete for $\mathcal{R}_{\mathcal{P}}$.

The evaluation problem $E(R_{CIRC})$ is the standard $\mathcal{P}$-complete problem [23]. $E(R_{CIRC})$ has been reduced to a large number of other problems in $\mathcal{P}$, thus showing that these problems are also $\mathcal{P}$-complete [21]. In many cases a logspace reduction from $E(R_{CIRC})$ to a new problem L leads to a prediction-preserving reduction of R_{CIRC} to a fairly natural prediction problem that is related to L.

In the following theorem we list prediction problems that were found this way. We omit the details of the reductions but only mention the related $\mathcal{P}$-complete problems.

Theorem 21 *R_{CIRC} reduces to the prediction problems $R_{\emptyset CFG}$, $R_{POLYTOPE}$, (R_{HC}, c), (R_{HC}, c') (defined below), and each of these problems is prediction-complete for $\mathcal{R}_{\mathcal{P}}$.*

Augmented CFG emptiness: $R_{\emptyset CFG} = \{r \in \Gamma^* :$ r encodes a CFG $\}$, where $c(r)$ is the set of all collections of productions that when added to the grammar represented by r, yields a context free grammar that generates the empty language.

Convex polytope intersection:
$R_{POLYTOPE} = \{r \in \Gamma^* : r$ encodes a system of linear equations (coefficients are integers encoded in unary) that define a convex polytope contained in the unit cube of some dimension $n\}$, where $c(r)$ is the set of subcubes of the n-dimensional unit cube that have non-empty intersection with the polytope represented by r.

The last two prediction problems are defined with respect to the same set of representations. Let R_{HC} be the set of representations $\{r \in \Gamma^* : r$ encodes a conjunction of Horn clauses$\}$. Then the prediction problems are:

Horn clause consistency: The pair (R_{HC}, c), where $c(r)$ is the set of collections of facts that are consistent with the conjunction represented by r.

Horn clause implication: The pair (R_{HC}, c'), where $c'(r)$ consists of all single clauses that are logically implied by the conjunction represented by r.

$R_{\emptyset CFG}$ is related to the $\mathcal{P}$-complete problem of deciding for a given CFG G, whether $L(G) = \emptyset$ [23]. $R_{POLYTOPE}$ is related to Linear Programming which is $\mathcal{P}$-complete [9]. The predictability of the following close variant of $R_{POLYTOPE}$ is an open question.

Convex polytope membership:
$R_{CONVEX} = \{r \in \Gamma^* : r$ represents a system of linear equations$\}$, and $c(r)$ is the set of points inside the convex polytope represented by r.[5]

Horn clause consistency and Horn clause implication are related to the $\mathcal{P}$-complete problem of deciding whether the empty clause can be deduced from a given set of Horn clauses [23]. Angluin [1] shows the prediction-completeness of Horn clause implication using the methodology introduced here.

6 Relaxations of Definitions

For ease of presentation, we have used one fixed alphabet (Σ) for concepts, and one fixed alphabet (Γ) for representations. It is easy to generalize our definitions to the case where each concept class and each representation is over a different finite alphabet. In the most general case, a concept is simply some subset of some domain X, and a representation is an element of a set R such that each $r \in R$ denotes some subset of X, and there are general notions of length for the elements of X and of R. For example, if one wishes to ignore issues of precision in a continuous domain, a representation might involve a collection of real numbers, where the length of each number would be counted as one unit. In a more general definition of prediction, these new length measures would replace the length measure "number of characters" that we have used.

When attempting to predict the concept (language) accepted by an unknown automaton, it may be desirable to allow the prediction algorithm time polynomial in other reasonable parameters of the unknown representation (and induced concept). For example, it is reasonable to allow at least as much time for prediction as would be required for the evaluation problem. Note however that all of the representations we have considered have polynomial-time evaluation problems (i.e., are in $\mathcal{R}_{\mathcal{P}}$).

Our definition of a prediction-preserving reduction is not the most general definition that preserves polynomial predictability. The definition may be relaxed in a number of ways, including allowing $w \in c(r)$ iff $f(w, s, m) \notin c(g(r, m))$ in part (1) of Definition 7

[5]Note that there are several versions of this prediction problem, depending on whether the coefficients and/or the points are encoded in unary or binary.

(see [25]), allowing ϵ as a parameter to the mappings f and g (and $\frac{1}{\epsilon}$ as a parameter to t and q), or even allowing a reduction similar to a randomized Turing reduction, as opposed to the many-one reduction presented here.

Note that our proofs of completeness are not very sensitive to the particular definition of distribution-independent predictability we have chosen.

A non-probabilistic model of polynomial time predictability could be defined based on the "on-line" prediction model introduced in [25]. In that model the prediction algorithm is given an unbounded sequence of (unlabelled) words. The algorithm makes a prediction (as to whether the word is in the unknown concept) after receiving each word, and is told whether it is correct before receiving the next word. The mistake bound of an algorithm for a prediction problem R is the number of wrong predictions (in the worst case) for any representation $r \in R^{[s]}$ and sequence of words from $\Sigma^{[m]}$, expressed as a function of s and m.

A prediction problem R is polynomially predictable in this model if there is a polynomial time algorithm whose mistake bound grows polynomially in s and m. The transformations discussed in [4,25] imply that if a prediction problem is polynomially predictable in this mistake-bounded model, then it is also polynomially predictable according to the definition of Section 3. Also, there are prediction problems that are polynomially predictable in our model, but are not in the mistake-bounded model (probabilistic approximation is easier than worst case). The theorems of this paper concerning polynomial reducibility and completeness also hold for the above notion of polynomial predictability. In particular, the existence of 1-way functions implies that there is no polynomial time prediction algorithm whose mistake bound grows polynomially in s and m for any prediction problem that is prediction-complete for $\mathcal{R}_{\mathcal{P}}$.

7 Conclusion

We have shown that there is a close relationship between predictability problems and evaluation problems, and have given natural prediction-complete problems for each of a number of complexity classes. Our results suggest that an algorithm for predicting DFAs from examples alone may be very difficult to find, since R_{DFA} is prediction-complete for $\mathcal{R}_{\mathcal{LOG}}$.

The apparent difficulty of DFA prediction can be extended to other problems. Ehrenfeucht and Haussler [10] give a $n^{\log n}$ algorithm for learning (and hence predicting) decision trees. A natural open problem suggested by Haussler [16] is the problem of learning "strictly ordered decision graphs" - the layered graph analog of a decision tree, but with the additional constraint (simplifying the problem) that all queries within the i^{th} layer be a query on the variable x_i. This type of rule is the least generalization of the problem of learning strictly ordered decision trees, which was solved in [17]. However, strictly ordered decision graphs are essentially "unrolled" DFAs, and our results imply that if strictly ordered decision graphs (even with in-degree two) can be predicted, then so can all prediction problems in $\mathcal{R}_{\mathcal{LOG}}$.

There are a large number of open problems: For restricted types of formulas, in particular for DNF, how hard is the prediction problem? For what set of prediction problems is R_{DNF} prediction-complete? Is DNF prediction equivalent to prediction of all boolean formulas? Is boolean formula prediction equivalent to DFA prediction? Perhaps by relaxing our notion of reduction as discussed earlier, hardness results for these problems may be found. A main open problem is to extend the hardness results given in this paper, i.e., to extend our list of problems that are prediction-complete for $\mathcal{R}_{\mathcal{P}}$.

Assuming the existence of appropriate 1-way functions, can one construct prediction problems that are not predictable and whose evaluation problems lie in lower complexity classes such as $\mathcal{NC}^2$, $\mathcal{LOG}^2$ space or even $\mathcal{LOGCFL}$? What are complete prediction problems for $\mathcal{R}_{\mathcal{NC}^2}$ and $\mathcal{R}_{\mathcal{LOG}^2}$, and how low in the complexity hierarchy does one need to go to ensure predictability?

Finally, Angluin [2] gives an algorithm for learning DFAs from examples and *membership queries*. What is the appropriate notion of reduction when the prediction algorithm is allowed to make queries about particular words? If Corollary 15 can be proved with such an extended notion of reduction, then it can be used to show that all boolean formulas are predictable by an algorithm that is allowed to make membership queries.

Acknowledgements

We thank Dana Angluin, David Haussler, Nick Littlestone, Michael Kearns and Allen VanGelder, for helpful discussions concerning this work. We also thank Ron Rivest for suggesting the use of the random functions of [13] to show that learning or prediction problems are hard.

References

[1] D. Angluin. *Learning Propositional Horn Sentences with Hints.* Technical Report YALEU/DCS/RR-590, Department of Computer Science, Yale University, December 1987.

[2] D. Angluin. *Learning Regular Sets From Queries and Counterexamples.* Technical Report YALEU/DCS/TR-464, Department of Computer Science, Yale University, March 1986.

[3] D. Angluin. On the complexity of minimum inference of regular sets. *Inform. Contr.*, 39(3):337–350, 1978.

[4] D. Angluin. Queries and concept learning. *Machine Learning*, 2, 1987.

[5] D. Beaver. *Polynomially sized boolean circuits are not learnable.* Technical Report TR-13-87, Aiken Computation Laboratory, Harvard University, December 1987.

[6] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. Classifying learnable geometric concepts with the Vapnik-Chervonenkis dimension. In *Proceedings of the 18th Annual ACM Symposium on Theory of Computation*, pages 273–282, Assoc. Comp. Mach., Berkeley, California, May 1986.

[7] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. *Learnability and the Vapnik-Chervonenkis Dimension.* Technical Report UCSC-CRL-87-20, Department of Computer and Information Sciences, University of California, Santa Cruz, November 1987. To appear, *J. ACM.*

[8] S. R. Buss. The Boolean formula value problem is in Alogtime. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computation*, pages 123–131, Assoc. Comp. Mach., New York, New York, May 1987.

[9] D. Dobkin, R. J. Lipton, and S. Reiss. Linear programming is Log-Space hard for P. *Inform. Process. Letters*, 8(2):96–97, 1979.

[10] A. Ehrenfeucht and D. Haussler. *Learning Decision Trees from Random Examples.* Technical Report UCSC-CRL-87-15, Department of Computer and Information Sciences, University of California, Santa Cruz, October 1987.

[11] A. Ehrenfeucht, D. Haussler, M. Kearns, and L. G. Valiant. *A general lower bound on the number of examples needed for learning.* Technical Report UCSC-CRL-87-26, Department of Computer and Information Sciences, University of California, Santa Cruz, 1987.

[12] E. M. Gold. Complexity of automaton identification from given data. *Inform. Contr.*, 37:302–320, 1978.

[13] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.

[14] M. A. Harrison. *Introduction to Formal Language Theory.* Addison-Wesley, Reading, Massachusetts, 1978.

[15] D. Haussler. *Learning Conjunctive Concepts in Structural Domains.* Technical Report UCSC-CRL-87-01, Department of Computer and Information Sciences, University of California, Santa Cruz, February 1987. To appear in *Machine Learning.*

[16] D. Haussler. Private communication. 1987.

[17] D. Haussler. Space efficient learning algorithms. 1986. Unpublished Manuscript.

[18] D. Haussler, M. Kearns, N. Littlestone, and M. K. Warmuth. *Equivalence of Models for Polynomial Learnability.* 1987. Technical report in preparation, Department of Computer and Information Sciences, University of California, Santa Cruz.

[19] D. Haussler, N. Littlestone, and M. K. Warmuth. Predicting {0,1} functions on randomly drawn points. Technical report in preparation, Department of Computer and Information Sciences, University of California, Santa Cruz.

[20] D. Haussler and E. Welzl. Epsilon-nets and simplex range queries. *Discrete Computational Geometry*, 2:127–151, 1987.

[21] H. J. Hoover and W. L. Ruzzo. *A Compendium of Problems Complete for P.* 1985. Unpublished manuscript, Department of Computer Science, University of Washington.

[22] N. D. Jones. Space-bounded reducibility among combinatorial problems. *J. Comp. Sys. Sci.*, 11:68–85, 1975.

[23] N. D. Jones and W. T. Laaser. Complete problems for deterministic polynomial time. *Theoretical Computer Science*, 3:105–117, 1977.

[24] M. Kearns, M. Li, L. Pitt, and L. G. Valiant. On the learnability of boolean formulae. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computation*, Assoc. Comp. Mach., New York, New York, May 1987.

[25] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1987.

[26] N. Lynch. Log space recognition and translation of parenthesis languages. *J. ACM*, 24(4):583–590, 1977.

[27] L. Pitt and L. G. Valiant. *Computational Limitations on Learning From Examples.* Technical Report TR-05-86, Aiken Computation Laboratory, Harvard University, July 1986. To appear, *J. ACM.*

[28] I. H. Sudborough. On tape-bounded complexity classes of multihead finite automata. *J. Comp. Sys. Sci.*, 10:62–76, 1975.

[29] L. G. Valiant. A theory of the learnable. *Comm. Assoc. Comp. Mach.*, 27(11):1134–1142, 1984.

[30] V. N. Vapnik and A. Ya. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Th. Prob. and its Appl.*, 16(2):264–280, 1971.