Prediction-Preserving Reducibility

Leonard Pitt*
Department of Computer Science
University of Illinois
Urbana, Illinois 61801

Manfred K. Warmuth**
Department of Computer and
Information Sciences
University of California
Santa Cruz, California 95064

Abstract

We investigate a model of polynomial-time concept prediction [30, 31] which is a relaxation of the distribution-independent model of concept learning due to Valiant [53]. *Prediction-preserving reductions* are defined, and are shown to be effective tools for comparing the relative difficulty of solving various prediction problems. A number of prediction-preserving reductions are given. For example, if deterministic finite automata are polynomially predictable, then so are all Boolean formulas.

We develop a complexity theory for prediction problems that parallels standard complexity theory. It is shown that certain problems of concept prediction are "prediction-complete" for a complexity class — a polynomial time algorithm for the prediction problem would imply that all languages in the complexity class are polynomially predictable. For example, polynomial-time prediction of deterministic finite automata implies the polynomial predictability of all languages in the class $\mathcal{LOG}$ (deterministic logspace). Similar natural prediction-complete problems are given for the standard complexity classes $\mathcal{NC}^1, \mathcal{NLOG}, \mathcal{LOGCFL}$, and $\mathcal{P}$. Showing that a prediction problem is prediction-complete for any of these classes provides varying degrees of evidence that no efficient prediction algorithm exists for the problem.

Based on very weak cryptographic assumptions, we establish hardness results for prediction of Boolean circuits and other prediction problems that are prediction-complete for $\mathcal{P}$. The recent related results of Kearns and Valiant [39] are discussed, which show that Boolean formulas and DFAs are not polynomially predictable based on the assumed intractability of computing specific cryptographic functions.

*Supported in part by NSF grant IRI-8809570, and by the Department of Computer Science, University of Illinois at Urbana-Champaign.

**Supported in part by ONR grant N00014-86-K-0454. Part of this work was done while visiting the Aiken Computation Laboratory, Harvard University, Cambridge, MA 02138, with partial support from ONR Grant N00014-85-K-0554.

1 Introduction

In this paper we are concerned with the learning of *concepts* from examples. Imagine a domain X of possible real world observations. Intuitively, a concept c is simply a partition of the observations into *positive examples* and *negative examples.* A learning algorithm is given randomly generated positive and negative examples of some unknown *target* concept c to be learned, and must produce a concept c' (a *hypothesis*), such that it is unlikely that c and c' will disagree on the classification of a new randomly generated example.

This extensional definition of a concept is not particularly useful, and in practice, it is desirable to assume that each concept c has associated with it a description r in some given *representation* language. Given any class of possible representations R, there is an associated *concept class* C, consisting of concepts described by elements of R. For example, Boolean formulas and DFAs (deterministic finite automata) are classes of representations that induce the concept classes of Boolean functions and regular languages, respectively. Henceforth we interchangeably use R to refer to a class of representations, as well as the concept class that it induces, and similarly, $r \in R$ will denote either the representation or the concept represented. As we discuss below, the representational power associated with a given concept description language R is a significant factor in determining whether R may be efficiently learned.

Distribution independent learning

The definition of distribution independent learnability (called *pac-learnability* [5]) of Valiant [53] addresses these representational issues, among others. A concept class R over domain X is pac-learnable if there exists a polynomial-time algorithm A such that for any $r \in R$, if A is given randomly generated elements of X (chosen according to an arbitrary, unknown distribution D on X), and told which are positive and which are negative examples of (the concept described by) r, then in polynomial time, A will output, with high probability (at least $1-\delta$) a concept (description) $r' \in R$ for which the concept r' approximates the target concept r in the following sense: the probability that an example generated according to D will be classified differently by r and r' is at most ϵ.

The values δ and ϵ are given as parameters to the algorithm A, reflecting the desired degrees of *confidence* $(1-\delta)$ in the performance of A, and the *accuracy* $(1-\epsilon)$ of the hypothesis r' produced by A. The run time of A, and hence the number of examples seen, may grow at most polynomially in $\frac{1}{\epsilon}$, $\frac{1}{\delta}$, as well as in other natural parameters reflecting the complexity of the learning task (for example, allowing the time to depend on the length of the description r of the target concept is typical). The algorithm is required to work regardless of the distribution D on the domain X.

A main goal in computational learning theory is to determine, for various definitions of successful learning, and in particular for pac-learnability, those concept classes that may be (efficiently) learned from examples. While a number of learnability results have been given (the papers [7, 11, 29, 37, 38] provide partial surveys), the learnability or nonlearnability of many natural classes remains undetermined. For example, the learnability of concept classes defined by Boolean formulas, restricted types of Boolean formulas (e.g., formulas in disjunctive normal form (DNF)), DFAs[1], NFAs, PDAs, CFGs, and Turing machines, remains unresolved.

In the search for learnable classes, a variant of pac-learning has been introduced: allow the algorithm A to output a description of the target concept, chosen from a *different* description language. Thus, following [45], we say that (the class of representations) R is *pac-learnable in terms of* (the class of representations) R', if the definition of pac-learnability above holds, but the concept description output by A is an element of R'. The most general relaxation possible is to that of (polynomial-time) *prediction*, where A is not required to output a description at all, but must arrive at a state in which it can predict (classify) future examples accurately (i.e., with error at most ϵ) with respect to the target concept. This notion of polynomial predictability, formally defined in Section 2, was introduced in [30, 31]. The polynomial predictability of R is equivalent to the existence of *any* class R' and algorithm A such that A pac-learns R in terms of R' [30].

A useful tool for studying pac-learnability (and predictability) is the Vapnik-Chervonenkis (VC) dimension [11, 32, 54] of a concept class. The VC dimension is a combinatorial parameter of a concept class with the following property. Let R be a concept class, and let R_s consist of all elements of R of length s. If the VC dimension of R_s grows more than polynomially in s, then R is not polynomially predictable (and hence not pac-learnable) [11, 19, 31]. In contrast, if the VC dimension of R_s grows polynomially in s (which is the case for all classes considered in this paper), a number of techniques have been given for finding learning and prediction algorithms with good performance, that cannot necessarily be implemented efficiently. Such techniques include: finding "small" consistent hypotheses [11, 12]; finding a prediction algorithm with small *permutation index* [31]; or finding a prediction algorithm with a polynomial worst case mistake bound [43]. Unfortunately, for the problems we consider in this paper, efficient implementations of these techniques have not been found.

Previous (partially) negative results

In the absence of positive learnability or predictability results, we hope to show nonlearnability or nonpredictability. For a class R, let the *consistency problem* for R be the problem

[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 [3].

of determining, given a collection of positive and negative examples, whether there exists an $r \in R$ that is *consistent* with the examples, i.e., classifies the examples correctly. As shown in [45] (see also [11, 30]), if the consistency problem for R is $\mathcal{NP}$-hard, then assuming that $\mathcal{RP}$ (random polynomial time) is not equal to $\mathcal{NP}$, R is not pac-learnable.

For example, it has been shown that for each constant $k \geq 2$, the class of k-term DNF formulas is not pac-learnable (assuming $\mathcal{RP} \neq \mathcal{NP}$) [45]. This is true because for each $k \geq 2$, it is $\mathcal{NP}$-hard to determine whether there exists a k-term DNF expression consistent with given examples. While this shows that k-term DNF is not learnable, it *does not* show that k-term DNF is not learnable in terms of some other representation. In fact, as was pointed out in [45], k-term DNF is pac-learnable in terms of k-CNF (Boolean formulas in conjunctive normal form with at most k literals in each clause). Similarly, it may be possible that k-term DNF is pac-learnable in terms of $f(k)$-term DNF for some polynomial f, and it would follow that DNF is pac-learnable [12]. Consequently, this type of non-learnability result relies on the syntactic constraints imposed by the requirement that the hypothesis of the algorithm must be expressed in some particular representation (e.g., k-term DNF, for some particular k). If these constraints are relaxed (e.g., to k-CNF), then there may be pac-learning algorithms (e.g., k-term DNF is pac-learnable in terms of k-CNF).

Similarly, it has been shown [45] that Boolean threshold functions[2] are not pac-learnable unless $\mathcal{RP} = \mathcal{NP}$. However, it is easy to learn Boolean threshold functions in terms of halfspaces by using linear programming [11], or by using the algorithm Winnow of [43]. Haussler [27] gives similar nonlearnability results that rely on representational constraints.

The research reported in this paper was motivated by attempting to determine the complexity of learning or predicting concept classes defined by DFAs and other types of automata. (See [44] for a survey on DFA learning.) By work of Blumer, Ehrenfeucht, Haussler, and Warmuth [12], and Board and Pitt [13], the pac-learnability of DFAs is equivalent to the existence of a randomized *Occam algorithm* for the *minimum consistent DFA problem*. Such an Occam algorithm takes as input any collection of strings (labeled "accept" or "reject") and produces (with high probability) a DFA that is consistent with the labeled strings, and whose size is bounded by the product of (1) a polynomial in the size of the smallest consistent DFA, and (2) a fractional power of the number of strings in the sample. Thus the pac-learnability of DFAs hinges on showing that the minimum consistent DFA problem can be very weakly approximated in random polynomial time.

Since the minimum consistent DFA problem is intimately related to the pac-learnability of DFAs, this problem has received significant study. Gold [21] shows that it is $\mathcal{NP}$-hard to find the smallest DFA consistent with a given sample. Angluin [2] shows that this is true even if all but ϵ of the words up to a given length are given as examples. Li and

[2]Such a function is given by a clause and a threshold k. The function is true for all assignments that make at least k literals in the clause true.

Vazirani extend the $\mathcal{NP}$-hardness result of [21] by showing that it is $\mathcal{NP}$-hard to produce a consistent NFA that is at most $\frac{9}{8}$ times larger than the smallest consistent DFA [42]. In [46], we show (assuming $\mathcal{P} \neq \mathcal{NP}$) that there is no polynomial-time approximation algorithm that is guaranteed to produce a consistent NFA of size bounded above by *any* polynomial in the size of the smallest DFA.

Although these results show that finding small DFAs consistent with a given sample is difficult, even the last result mentioned does not rule out the possibility of the existence of an Occam algorithm for DFAs, since such an algorithm would be allowed to produce a consistent DFA whose size depended not only polynomially in the smallest consistent one, but also could be as large as a fractional power of the number of examples in the input sample. The problem of extending the results of [46] to show that no Occam algorithm exists seems very difficult. Furthermore, even if such a result were obtained, showing that DFAs were not pac-learnable, it would still be possible that DFAs were pac-learnable in terms of some other class of representations (not necessarily of the regular sets), and hence DFAs would be polynomially predictable.

Thus, we have not been able to show that a number of concept classes are learnable, and, on the other hand, the partial nonlearnability results for these problems rely on syntactic constraints that reflect the complexity of the consistency problem associated with a given choice of representations[3]. As discussed above, polynomial predictability captures learnability in terms of an arbitrary hypothesis class [30]. Thus, negative results for predictability are more meaningful; they reflect the complexity of noticing patterns in the data, as opposed to simply reflecting the syntactic difficulties of expressing such patterns. Negative results for prediction imply negative results for learning.

A complexity theoretic approach

The theory of complexity classes and reducibilities (e.g., $\mathcal{NP}$-completeness) has been particularly useful in providing evidence for the intractability of computational problems. Here we develop a similar complexity theory for predictability. 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 unlabeled examples of the first problem to unlabeled examples of the second problem, and a function g that maps representations

[3]In a more demanding, deterministic model of learnability, Angluin has shown that DFAs are not learnable by polynomially many *equivalence queries* [6], but are learnable by polynomially many equivalence and *membership* queries [3]. Neither of these results has any implications in the model considered here, where examples are generated according to an arbitrary probability distribution.

of the first problem to representations 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 similar to the ones given in [37, 43], except that we allow a variety of domains, and we determine general sufficient conditions that ensure the preservation of polynomial-time predictability. Littlestone [43] gives reductions between prediction problems in Boolean domains for a different model of predictability discussed in Section 8. Kearns *et al* [37] give reductions between various pac-learning problems in the Boolean domain, and Haussler [27] gives reductions between pac-learning problems in structural domains.

We review reductions from previous work that have been given in a less formal setting, and present as examples a number of new reductions. One such reduction shows that predicting arbitrary Boolean formulas is no easier even if the prediction algorithm is given the description of a tree circuit that computes the formula, with all gates specified, except the mapping of variables to inputs of the circuit is unknown.

With each prediction problem, we associate an *evaluation problem*: given a string and a representation, is the string a positive or a negative example of the concept denoted by the representation? We classify prediction problems by the complexity of their evaluation problems. This gives rise to a notion of *prediction-completeness*; if a prediction problem is prediction-complete for a given complexity class, then its predictability implies the predictability of all prediction problems whose evaluation problem is in that complexity class.

Using these ideas, we give a prediction-preserving reduction from an arbitrary prediction problem whose evaluation problem is in deterministic logspace ($\mathcal{LOG}$) to the prediction problem for DFAs. This shows that the prediction problem for DFAs is prediction-complete for $\mathcal{LOG}$. Since determining whether an assignment satisfies a given Boolean formula is in $\mathcal{LOG}$, it follows that predicting arbitrary Boolean formulas reduces to predicting DFAs.

Analogous to the deterministic case, we show that the prediction problems for NFAs, CFGs, and alternating DFAs are prediction-complete for the complexity classes $\mathcal{NLOG}$,$\mathcal{LOGCFL}$, and $\mathcal{P}$, respectively, and we show that prediction of Boolean formulas is prediction-complete for the complexity class $\mathcal{NC}^1$.

Problems prediction-complete for polynomial time — hardness results

In addition to the alternating DFA prediction problem, we describe a number of other prediction problems that, if predictable, would imply the predictability of all languages accepted in polynomial time. Our list of such hard prediction problems includes

- Convex Polytope Intersection: the concept is an unknown convex polytope; positive examples are cubes that have non-empty intersection with the polytope.
- Horn Clause Consistency: the concept is an unknown conjunction of Horn clauses; positive examples are sets of facts that are consistent with the conjunction.
- Augmented CFG Emptiness: the concept is an unknown context free grammar; positive examples are sets of productions that when added to the grammar, yield a grammar generating the empty language.

It follows from the work of Goldreich, Goldwasser, and Micali [22] that these prediction problems are not predictable (even in an extremely weak sense) assuming the existence of cryptographically secure pseudorandom bit generators, which is equivalent to the existence of a certain type of one-way function [41].

Subsequent to this research, Kearns and Valiant [39] show that a polynomial-time learning or prediction algorithm for DFAs can be used to invert certain cryptographic functions. This is done by first showing that predicting arbitrary Boolean formulas is as hard as inverting the given cryptographic functions. Next, apply Corollary 5.7, which shows that prediction of Boolean formulas reduces to prediction of DFAs. Consequently, DFAs are not polynomially predictable based on the same cryptographic assumptions.

The rest of this paper is organized as follows. In Section 2 we formally define polynomial-time predictability. Section 3 introduces the notion of a prediction-preserving reduction, and gives a number of examples. In Section 4 we examine additional properties of prediction-preserving reductions. We associate a language (called the evaluation problem) with each prediction problem, and we define what is meant for a prediction problem to be prediction-complete for a standard complexity class. Section 5 gives some of our main results: we relate automata prediction to various complexity classes, and in particular, we show that DFAs are prediction-complete for $\mathcal{LOG}$. In Section 6 we describe some prediction problems that are as hard to predict as any language in polynomial time, and describe in Section 7 why it follows that these problems are not predictable based on certain cryptographic assumptions. In Section 7 we also discuss the results of [39] showing that DFAs are not predictable based on different cryptographic assumptions. Finally, in Section 8 we summarize our results, and discuss a number of interesting open problems.

2 Polynomial Predictability

Formal definitions for the complexity classes discussed in this paper may be found in [58], and in the references given below. $\mathcal{LOG}$, $\mathcal{NLOG}$, $\mathcal{P}$, and $\mathcal{NP}$ denote the classes of languages accepted in deterministic logspace, nondeterministic logspace, polynomial time, and nondeterministic polynomial time, respectively. $\mathcal{LOGCFL}$ denotes the class of languages accepted by polynomial time bounded auxiliary (nondeterministic) PDAs with logarithmic additional work tape [52]. $\mathcal{NC}^1$ denotes the class of U_{E^*}-uniform bounded fan-in circuit families of logarithmic depth (and hence polynomial size) [16, 48]. Ruzzo [48] showed that this is exactly the class of languages accepted by some *alternating* Turing machine in logarithmic time. (See [15, 48, 58] for definitions.)

Throughout the paper, let Σ and Γ be fixed, finite alphabets.

Definition 2.1 *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 2.2 *A* prediction problem *is a pair (R, c), where $R \subseteq \Gamma^+$, and c is a mapping $c : R \to 2^{\Sigma^*}$. R is a "set of representations", and each $r \in R$ denotes the concept $c(r) \subseteq \Sigma^*$. The concept class represented by (R, c) is $\{c(r) : r \in R\}$.*

Generally, given any R, the mapping c will be implicitly understood, hence we use R as an abbreviation for the prediction problem (R, c). We also use R to denote the concept class it represents.

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.

For ease of presentation, Σ was chosen as some fixed finite alphabet over which concepts are defined. Consequently, all formal language defined below are languages over Σ. As discussed in Section 8, our results hold without this restriction to a single alphabet. In any case, the restriction to a fixed alphabet is justified for the purposes of polynomial predictability (defined below), because the classes of languages considered are closed under homomorphisms.

- $R_{DFA} = \{r : r$ encodes a DFA$\}$ is a set of representations (for the concept class of regular languages over the fixed alphabet Σ) 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$\}$.

Define a Boolean formula over Boolean variables $\{x_1, \ldots, x_n\}$ as a (not necessarily binary) rooted tree with internal nodes labeled with elements of $\{\vee, \wedge, \neg\}$, leaves labeled with elements of $\{x_1, \ldots, x_n\}$, and such that internal nodes labeled with $\vee$ or $\wedge$ have at least two children, and internal nodes labeled with $\neg$ have exactly one child. The value of a Boolean formula is defined in the usual way as the value of the circuit it represents, with the leaves as inputs, the root as output, and the interior nodes labeled $\wedge$, $\vee$, and $\neg$ interpreted as Boolean gates for logical AND, OR, and NOT, respectively.

- $R_{BF} = \{r : r$ encodes a binary number n and a Boolean formula over variables $\{x_1, \ldots, x_n\}\}$. 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.
- $R_{DNF} = \{r : r$ encodes a number n in binary and a Boolean formula in disjunctive normal form over variables $\{x_1, \ldots, x_n\}\}$.
- $R_{CNF} = \{r : r$ encodes a number n in binary and a Boolean formula in conjunctive normal form over variables $\{x_1, \ldots, x_n\}\}$.
- $R_{CONVEX} = \{r : r$ encodes a number d in binary, and a system of linear inequalities over d variables (coefficients are integers encoded in binary) that defines a convex polytope$\}$. The concept $c(r)$ consists of all d-dimensional vectors that are solutions to the system of equations represented by r. The components of a vector are encoded in binary.

Before formally defining polynomial predictability, we must consider the amount of resources (time and number of examples) that should be available to the prediction algorithm. It is natural to allow the resources to grow polynomially in the inverse of the parameter ϵ, an upper bound on the desired predictive error of the prediction algorithm. As discussed above, the length (number of letters[4]) of the representation r of the unknown target concept $c(r)$ is a measure of its complexity; consequently, we will allow the resources to grow polynomially in a parameter s, which is an upper bound on $|r|$.

Finally, the resources of the prediction algorithm should be allowed to grow with the length of the input examples. For example, for prediction of Boolean formulas, the time (and number of examples) allowed will depend on n, the number of variables over which the formula is defined. However, for some concept classes, the words of a concept may not all have the same length (e.g., the concepts (languages) defined by DFAs, CFGs, etc.). An arbitrary probability distribution may supply words of significantly different lengths as examples. Because the words are chosen randomly, there are subtle issues involved in specifying in a natural way how the resources of the algorithm may depend on the word lengths.

We illustrate this difficulty by considering Angluin's approach regarding the same issue in a deterministic setting [6, 3]. In this model, the algorithm is allowed, at any point, time polynomial in the longest word that it has received up to that point. Suppose we adopt this definition in our stochastic setting, and consider a prediction algorithm that is allowed time (and hence a number of examples) at most quadratic in the length of the longest example yet seen. More specifically, a prediction algorithm using a quadratic number of examples may continue to run providing that the number of examples requested is less than the square of the length of the longest example yet seen, but must halt as soon as these values become equal.

We construct a distribution for which such an algorithm may never halt, although it is quadratically resource bounded. Let the distribution be such that for each $i \geq 1$, exactly one word of length 2^i occurs with probability 2^{-i}. Observe that if the algorithm halts, then it does so right after requesting the 2^{2i}-th example, for some $i \geq 1$, and all examples seen to that point are of length at most 2^i. The examples of length at most 2^i have total probability $1 - 2^{-i}$. Thus the probability of stopping right after trial 2^{2i} is at most $(1-2^{-i})^{2^{2i}} < e^{-2^i} \leq e^{-2i}$. Thus the probability of ever stopping is at most $\frac{e^{-2}}{1-e^{-2}}$, which is less than $\frac{1}{6}$.

This example shows that, depending on the distribution, a resource bounded prediction algorithm may have significant probability of never halting (and therefore failing). However, we would like our definition of predictability to require that successful prediction be

[4]Other measures of length are discussed in Section 8.

achievable with any arbitrarily small probability.

Another approach is to allow the time used by the prediction algorithm to depend on the variance and the expectation of the length of an example. Observe that in the above example the expected length is infinite, so again a resource bounded algorithm would be allowed infinite time.

Perhaps the most natural way to deal with this issue is to supply the algorithm with parameters n and γ, where the probability that a word of length greater than n occurring is at most γ. The resources available to the prediction algorithm would then depend on n and the inverse of γ. For simplicity, we take a related approach; we assume that for some n, the probability of example words of length greater than n is 0. The prediction algorithm is given n as a parameter and the resources of the algorithm are allowed to grow polynomially in this parameter as well. Note that for any distribution D on the countably infinite space Σ^*, and for any arbitrarily small $\gamma > 0$, there is a length n such that the probability of any word of length greater than n occurring is at most γ. Thus there is a distribution D' assigning zero probability to words of length greater than n such that D' "approximates" D within γ. Consequently, our restriction to such distributions is not an unreasonable one.

Definition 2.3 *For any language L, let $L^{[n]} = \{w \in L : |w| \leq n\}$.*

Definition 2.4 *For any representation r (and induced concept $c(r)$), and for any word w, let $label_{c(r)}(w)$ = "+" if $w \in c(r)$ and "−" if $w \notin c(r)$. An* example *of $c(r)$ is a pair $\langle w, label_{c(r)}(w)\rangle$. An* unlabeled *example is just a word w.*

Definition 2.5 *A* prediction algorithm *A is an algorithm[5] that takes as input three parameters s, n, and ϵ, a collection of elements of $\Sigma^{[n]} \times \{+,-\}$, and an element $w \in \Sigma^{[n]}$. 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 $t(s, n, \frac{1}{\epsilon}, l)$, where l is the total length of the input of A.*

Definition 2.6 *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, n, and $\epsilon > 0$, for all $r \in R^{[s]}$, and for all probability distributions on $\Sigma^{[n]}$, if A is given at least $p(s, n, \frac{1}{\epsilon})$ randomly generated examples of (the unknown* target concept*) $c(r)$, and a randomly generated unlabeled word $w \in \Sigma^{[n]}$, then the probability that A incorrectly predicts $label_{c(r)}(w)$ is at most ϵ.*

[5]In this paper we consider only *deterministic* algorithms; all probabilities are taken over the distribution on examples. Our results also hold in the case of probabilistic prediction and learning algorithms. Relationships between such deterministic and probabilistic algorithms are discussed in [30].

Comments: From here on, "predictable" means "polynomially predictable". Our model is based on the definition of predictability introduced in [31]. (In this paper we also allow the number of examples to grow with n for the reasons discussed above.) These definitions of predictability are analogous to the pac-learnability introduced by Valiant [53]; the correspondence is as follows: Let $error(A)$ denote the probability that A predicts incorrectly on the unlabeled word. Predictability requires that $error(A) \leq \epsilon$. As outlined in the Introduction, for pac-learning the algorithm must, with probability at least $1 - \delta$, 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 (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 [30].

Polynomial predictability is not sacrificed if the parameters s, n, and ϵ are not explicitly given to the algorithm. This may be seen by an argument similar to the one used in [30]. Suppose that prediction algorithm A and polynomial p witness that the prediction problem R is predictable as required by Definition 2.6. Although A receives as input the parameters s, n, and ϵ, we construct B that uses no parameters, and satisfies Definition 2.6.

Assume without loss of generality that p is increasing in each of its three arguments. Define $\hat{p}(x) = p(x, x, x)$. On input of u examples, B computes the largest integer x such that $\hat{p}(x) \leq u$. If x is greater than or equal to the length of the longest example, then B simulates A with the parameters $s = x, n = x$, and $\epsilon = \frac{1}{x}$ on the first $\hat{p}(x)$ examples, and predicts as A does on the unlabeled example. Otherwise, x is smaller than the length of the longest example, and B simply predicts $+$ on the unlabeled (test) instance.

If B is given $u \geq \hat{p}(s + n + \frac{1}{\epsilon})$ examples of length at most n of some unknown concept of size at most s, then the value x computed is at least $s + n + \frac{1}{\epsilon}$. In particular, $x \geq n$, the length of the longest example, and therefore B simulates A with the three parameters x, x, and $\frac{1}{x}$. The first two parameters exceed s and n, and the last is at most ϵ. The simulation uses $\hat{p}(x) = p(x, x, x)$ labeled examples. Consequently, the probability that A (and hence B) is incorrect in predicting the unlabeled example is at most $\frac{1}{x} \leq \epsilon$.

3 Prediction-Preserving Reductions

We are now ready to define prediction-preserving reductions among prediction problems. The intent of such a reduction is that it preserves polynomial predictability. 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 3.1 *Let R and R' be prediction problems (with implicit mappings c and c', respectively). Then R reduces to R' (denoted by $R \trianglelefteq R'$) iff there are functions $f : \Sigma^* \times \mathbb{N} \times \mathbb{N} \to \Sigma^*$ (the word transformation) and $g : R \times \mathbb{N} \times \mathbb{N} \to R'$ (the representation transformation), and polynomials t and q such that for all $s, n \in \mathbb{N}$, $r \in R^{[s]}$, and $w \in \Sigma^{[n]}$,*

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

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

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

If $R \trianglelefteq R'$ and $R' \trianglelefteq R$ then we write $R \cong R'$.

Comments: The definition does not quite fit the intuitive description above, in that the word transformation f and the representation transformation g both have additional parameters s and n. In most simple cases, these additional parameters are unnecessary, and transformations f and g such that $w \in c(r)$ iff $f(w) \in c'(g(r))$ will be sufficient. However, in some cases, no such simplified reduction seems possible, and it will be necessary for the transformation f to depend also on s, a bound on $|r|$ (the length of the representation of the unknown target concept), and on n (a bound on the lengths of words for which the reduction is applicable). Similarly, g will also depend on its additional parameters. (Although g is given r as a parameter, the upper bound s on $|r|$ is also needed to ensure transitivity of the reducibility (Lemma 4.1) in the case that s is more than polynomially larger than $|r|$. This corrects earlier versions of this manuscript where s was not included as a parameter to g.)

Requirement (1) specifies that the word transformation must map positive examples to positive examples. A straightforward modification (as in [43]) allows the following condition as an alternative.

(1') $w \in c(r)$ iff $f(w, s, n) \notin c'(g(r, s, n))$.

In all but one of our reductions, requirement (1) is sufficient. All proofs will assume that f and g satisfy (1) instead of (1'); only minor modifications are necessary for any theorem presented herein if (1') is allowed also.

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 the definition of a prediction-preserving reduction in Section 8.

In our discussion below, we assume that prediction-preserving reductions do indeed preserve predictability, i.e, that if $R \trianglelefteq R'$, then the predictability of R' implies the predictability of R, and thus the prediction problem R' is at least as hard as the prediction problem R. The proof of this property appears in Section 4 (Lemma 4.2). In the remainder of this section we exhibit some example reductions.

Example reductions

Let k-term DNF be the class of Boolean formulas in disjunctive normal form with at most k terms (i.e., disjuncts of at most k conjuncts), and let k-CNF be the class of Boolean formulas in conjunctive normal form, in which each clause has at most k literals (k-CNF). As pointed out in [45], for every k-term DNF formula over n variables, there is a logically equivalent k-CNF formula over the same n variables of size $O(n^k)$. For each constant k, this immediately gives a prediction-preserving reduction from k-term DNF to k-CNF, where f is the projection function of its first argument (i.e., f is the identity on w), and g maps a k-term DNF expression to the equivalent k-CNF expression.

A trivial prediction-preserving reduction that exemplifies the use of the alternate requirement (1') exists between CNF and DNF (in either direction), and thus $R_{CNF} \cong R_{DNF}$. The word transformation f is the identity on the assignment w, and the representation transformation g maps a DNF expression D into a CNF expression C (or vice-versa) such that C is equivalent to the negation of D. Note that there is no reduction for which f is the identity on w, and for which requirement (1) is satisfied (i.e., positive examples map to positive examples). This follows from the fact that there are DNF expressions for which the smallest equivalent CNF expression is exponentially larger.

A simple reduction shows that prediction of membership in convex regions is at least as hard as the prediction of CNF formulas. To see that $R_{CNF} \trianglelefteq R_{CONVEX}$, suppose r is the encoding of a CNF expression C over the variables $x_1, x_2, \ldots, x_n$. If $w = w_1 w_2 \cdots w_n$ is an n-bit assignment to the variables of C, then the word transformation f expands w into a $2n$-bit string $w\bar{w}$ containing the original bits followed by their negations (this trick was used in [37, 43]). Then the collection of inequalities $g(r, s, n)$ over the $2n$ variables $x_1, \ldots, x_n, x_{n+1}, \ldots, x_{2n}$ is constructed as follows. For each clause $\{a_{i_1}, a_{i_2}, \ldots, a_{i_k}\}$, where a_{i_k} is either x_{i_k} or its negation, include in $g(r, s, n)$ the inequality $b_{i_1} + b_{i_2} + \cdots + b_{i_k} \geq 1$, where $b_{i_j} = x_{i_j}$ if $a_{i_j} = x_{i_j}$, and $b_{i_j} = x_{n+i_j}$ if a_{i_j} is the negation of x_{i_j}.

As another example, observe that the language consisting of satisfying assignments (represented as a Boolean string of length n) of any DNF formula of n variables with s terms is accepted by an NFA of size $O(sn)$. (The NFA guesses which of s terms is satisfied, and branches to a chain of $O(n)$ states to verify that the n bit input satisfies the appropriate literals.) Thus predicting DNF trivially reduces to predicting NFAs, where again, f is the identity on w, and g maps the DNF expression to the corresponding NFA.

However, observe that the language consisting of satisfying assignments of a DNF formula might have an exponential size minimum DFA. For example, the satisfying assignments of the formula $x_1x_{\frac{n}{2}+1} + x_2x_{\frac{n}{2}+2} + \cdots + x_{\frac{n}{2}}x_n$, for n even, is a language whose smallest DFA has size at least $2^{\frac{n}{2}}$. Thus there is no prediction-preserving reduction from DNF to DFAs where f is the identity on w; some transformation must be applied to the set of satisfying assignments in order for the resulting language to be accepted by a DFA of size at most polynomially larger than the size of the DNF expression. We give such a reduction in Theorem 3.2 below.

A simple reduction in which f is not simply the identity on w is given in [37, 43]. A monomial is a conjunct of literals, i.e., a 1-term DNF formula. For any fixed k, the problem of predicting k-CNF formulas reduces to that of predicting monomials. By definition, each clause in a k-CNF formula has at most k literals. Thus there are $u = O(n^k)$ such clauses, where n is the number of variables. Let f map each n-bit assignment into a u-bit assignment, the i-th bit of which is 1 iff the i-th k-literal clause is 1. Given this transformation of the assignment, the mapping g simply expresses each k-CNF with v clauses as a monomial of size v over the enlarged variable set of size u. Note that since k is a constant, f is computable in time polynomial in n, and the size of the image of g is bounded by a polynomial in n.

In the above reductions, the transformation f did not depend on the parameter s, which is a bound on the length of the representation of the target concept in the original problem. The following reduction makes use of that parameter as well.

Theorem 3.2 $R_{DNF} \trianglelefteq R_{DFA}$.

Proof: Let $r \in R_{DNF}$ encode a DNF expression of n variables. Then each example assignment is a word of length n. The parameter s is a bound on the length of the target DNF expression r. In particular, s is an upper bound on the number of terms of r. For all assignments w, $f(w, s, n) = w^s$, i.e. f simply replicates w exactly s times. For a given DNF expression r with at most s terms it is easy to design a DFA A with $O(sn)$ states such that r is true on w iff A accepts w^s. In particular, for each of at most s terms, the DFA uses a chain of $O(n)$ states (and n input symbols) to check if the term is satisfied. If not, then A moves on to the next copy of w in the input, and the next set of states to test whether the next term is satisfied. Thus g simply needs to map r, s, and n into a representation r' of such an automaton A. For any reasonable encoding for DFAs we have that $|r'|$ is polynomial in sn. $\square$

In light of the apparent difficulty of predicting arbitrary Boolean formulas, Russell suggests [47] that it may be easier to predict certain restricted classes of Boolean formulas. Prediction-preserving reducibility is the appropriate tool to determine whether indeed, the prediction problems associated with these restricted classes are any easier than the unrestricted version. We extrapolate from his discussion and define two very restricted classes of Boolean formulas. Theorems 3.3 and 3.4 below show that R_{BF} reduces to both of these classes, and in fact that the polynomial predictability of both of these classes is equivalent to the polynomial predictability of R_{BF}, the unrestricted class of Boolean formulas. Hence it is unlikely that these restricted classes are predictable considering the results from [39] discussed in Section 7.

Recall that a Boolean formula is a Boolean tree, i.e., a Boolean circuit that is a tree, where each node is an $\wedge$-gate, an $\vee$-gate, or a $\neg$-gate, the input variables are at the leaves, and the value of the formula is computed at the root. Both restricted classes assume that the formula is a complete ordered binary tree. For the first class, the input variables $x_1, x_2, \ldots, x_n$ are provided (in that order) at the leaves, but the labels of the gates are unknown. For the second class, the labels of the gates are given, but the mapping of the variables to the leaves is unknown. Thus in the first case, the goal is to learn a formula when given as additional information the structure of the tree that computes it. In the second case, not only is the tree structure available, but all gate labelings as well — the only unknown is the mapping of input variables to leaves.

The two restricted classes are formalized by the following prediction problems. Note that we only use $\wedge$-gates and $\vee$-gates as labels. The height of a node is the length of the longest path to a leaf (leaves have height zero) and the height of a tree is the height of its root. The size of a tree T, denoted by $|T|$, is the number of nodes of T.

- $R_{TREE_1} = \{r : r$ encodes a complete ordered binary tree, where each internal node is either an $\wedge$-gate or an $\vee$-gate$\}$. The concept $c(r)$ consists of those Boolean strings w of length the number of leaves of r, such that when the bits of w are supplied (in order from left to right) at the leaves, the value computed at the root is 1.

- $R_{TREE_2} = \{r : r$ encodes a permutation π of $n = 2^k$ elements, for some $k\}$. For any k, let $T^{(k)}$ be the complete ordered Boolean tree of height k, where the gates at even height are $\vee$-gates and the gates at odd height are $\wedge$-gates. Then $c(r)$ consists of exactly those strings w of length 2^k, such that if the leaves of $T^{(k)}$ are labeled (in order from left to right) with the inputs $\pi(w)$, then $T^{(k)}$ evaluates to 1.

Theorem 3.3 $R_{TREE_1} \cong R_{BF}$.

Proof: The fact that $R_{TREE_1} \trianglelefteq R_{BF}$ follows almost immediately from the observation that every representation of a concept in R_{TREE_1} is also a representation of a Boolean formula, i.e., an element of R_{BF}. More formally, the reduction is witnessed by the transformations f and g as follows. The word transformation f is the identity on its first argument. Let $r \in R_{TREE_1}$ represent an ordered tree T. For $s \geq |r|$ and n at least as large as the number of leaves of $|T|$, define the representation transformation $g(r, s, n)$ to be a string consisting of the number n in binary followed by the representation r' of the Boolean formula defined by the tree T, where the i-th leaf from the left receives the input variable x_i. Clearly $|g(r,s,n)|$ is $O(|r|\log n)$ and thus g is polynomially length preserving.

It is somewhat more involved to show that $R_{BF} \trianglelefteq R_{TREE_1}$. Let F be a Boolean formula (a tree) over the variables $x_1, x_2, \ldots, x_n$. Let $|F|$ be the number of nodes in F. Using standard techniques [56], one can show that F can be computed by a Boolean circuit of depth at most $\lfloor b \log |F| \rfloor$, where b is a fixed constant independent of F. Also, by using two-railed logic, F can be computed by a monotone Boolean circuit of depth at most $\lfloor b' \log |F| \rfloor$ with the following properties. The input variables are $x_1, \ldots, x_n, \neg x_1, \ldots, \neg x_n$, all gates of the circuit have fan-in exactly two, and the circuit has only $\wedge$-gates and $\vee$-gates. Again, b' is a fixed constant independent of F. From this latter circuit it is easy to construct a binary tree circuit of height exactly $h = \lfloor b' \log |F| \rfloor$, with input variables $x_1, \ldots, x_n, \neg x_1, \ldots, \neg x_n$ that computes F (this will involve duplicating gates at each level).

In this tree circuit, all internal nodes have degree exactly two, and thus the tree has $l \leq 2^h$ leaves. If $l < 2^h$ (i.e., the tree is not complete), then we can pad the tree into a *complete* ordered binary tree without increasing the height. This is done by iteratively replacing "shallow" leaf nodes labeled with some literal z by a tree representing $z \wedge z$.

Let T_F denote the complete ordered binary tree constructed above, which computes the function F. T_F is not in the form required by R_{TREE_1}: Although T_F has 2^h leaves, the variables $x_1, \ldots, x_n, \neg x_1, \ldots, \neg x_n$ do not necessarily appear at the leaves of T_F in any particular order, and each may appear any number of times, depending on the transformations in the construction above. We construct a tree from T_F where the sequence of inputs that appear on the leaves (from left to right) follow a particular pattern. Let n' be the smallest power of 2 that is at least as large as n. We replace each leaf l in T_F by a complete ordered binary subtree such that:

1. The leaves of each subtree are labeled from left to right with the fixed input sequence
$$I = \langle x_1, 0, \ldots, x_n, 0, \neg x_1, 0, \ldots, \neg x_n, 0 \rangle \langle 0 \rangle^{4(n'-n)}.$$
2. The l-th subtree, replacing the l-th leaf, selects the function y_l, where y_l is the input of the l-th leaf in T_F.
3. The height of each subtree is $(\log n') + 2$.

Such subtrees can be built for each leaf using only $\wedge$-gates and $\vee$-gates. Denote the new tree that is constructed from T_F by $\widehat{T}_F$. Observe that $\widehat{T}_F$ has height $\lfloor b' \log |F| \rfloor + \lceil \log n \rceil + 2$.

We now give a reduction proving Theorem 3.3, relying on the above construction. For a given input string (assignment) w, let $I(w)$ be the string (assignment) resulting from substituting, for each i, the i-th bit of w in place of x_i in the sequence I above. The string transformation f is now defined by $f(w,s,n) = (I(w))^{2^{\lfloor b' \log s \rfloor}}$. The representation transformation g is defined as follows. If r is the encoding of a Boolean formula (a tree) F' over n variables, then clearly $|F'| \leq |r| \leq s$. Then F' can be padded into a tree F that computes the same function, and such that $|F| = s$. (The padding is as above — iteratively replace leaves labeled z by the subtree $z \wedge z$.) Now let $g(r,s,n) = \widehat{T}_F$, as defined above. (Technically, $g(r,s,n)$ is the representation of the tree $\widehat{T}_F$.)

By the above construction, $g(r,s,n) = \widehat{T}_F$ accepts $f(w,s,n) = (I(w))^{2^{\lfloor b' \log s \rfloor}}$ iff r accepts w. Clearly f is polynomial-time computable, and since $\widehat{T}_F$ has height $\lfloor b' \log s \rfloor + \lceil \log n \rceil + 2$, it follows that the number of nodes of $\widehat{T}_F$, and hence the size of its encoding, is at most a polynomial in s and n. $\square$

Theorem 3.4 $R_{TREE_2} \cong R_{BF}$.

Proof: As in the previous proof $R_{TREE_2} \trianglelefteq R_{BF}$ is the easier direction. Let f be the identity on its first argument. Let $r \in R_{TREE_2}$ represent a permutation π of size 2^k. Let T be the complete ordered Boolean tree of height k, where the gates at even height are $\vee$-gates and the gates at odd height are $\wedge$-gates and the i-th leaf from the left is labeled with $x_{\pi(i)}$, for $1 \leq i \leq 2^k$. For $s \geq |r|$ and $n \geq 2^k$ define the representation transformation $g(r,s,n)$ to be a string consisting of the number n in binary followed by the representation r' of the tree T. As in the previous proof, $|g(r,s,n)|$ is $O(|r| \log n)$ and thus g is polynomially length preserving. This concludes the proof of the first reduction: $R_{TREE_2} \trianglelefteq R_{BF}$.

We show in Section 4 that prediction-preserving reductions are transitive (Lemma 4.1). Thus, by Theorem 3.3, to prove $R_{BF} \trianglelefteq R_{TREE_2}$ it suffices to show that $R_{TREE_1} \trianglelefteq R_{TREE_2}$.

Let T be a complete ordered binary tree of height h with nodes labeled $\wedge$ and $\vee$, and with n leaves that receive the inputs $x_1, \ldots, x_n$. Let $h = \log n$ be the height of T. We embed T into a complete binary tree $\widehat{T}$ such that

1. $\widehat{T}$ computes the same function as T.
2. The height of $\widehat{T}$ is $2h+2$, the gates on all even levels are $\vee$ gates, and the gates on all odd levels are $\wedge$-gates.
3. The leaves of $\widehat{T}$ are labeled from left to right with a permutation π of

$$J = \langle x_1, \ldots, x_n \rangle^2 \langle 0, 1 \rangle^{2n^2 - n}$$

.

If for each T there exists a $\widehat{T}$ satisfying these conditions, the reduction can be easily completed as follows. For a given input string (assignment) w of length n, let $J(w)$ be the string (assignment) obtained by substituting, for each i, the i-th bit of w in place of x_i in the sequence J above. The string transformation f is defined as $f(w,s,n) = J(w)$. Let $r \in R_{TREE_2}$ represent the tree T. The representation transformation is given as $g(r,s,n) = \pi$, where π is as defined above. (Technically, $g(r,s,n)$ is the representation of π.) By definition, the concept $c(g(r,s,n)) = c(\pi)$ consists of those strings of length $4n^2$ that when permuted according to π, evaluate to 1 on the alternating $\wedge$-$\vee$ tree of height $\log 4n^2$. By construction, $f(w,s,n) = J(w)$ is such a string if and only if T evaluates to 1 on input w.

We now show how to construct $\widehat{T}$ from T. This is done by induction on the height h of T. Let E_0 and E_1 be two leaves with input 0 and 1, respectively. If T has height 0, then it consists of just one leaf L; make two copies L_1 and L_2 of that leaf and let $\widehat{T}$ be the following height 2 complete binary tree: $(L_1 \wedge E_0) \vee (L_2 \wedge E_1)$. Clearly $\widehat{T}$ computes the same function as T. Also $J = \langle x_1 \rangle^2 \langle 0, 1 \rangle$ in the case $h = 0$. Thus $\widehat{T}$ is labeled with a permutation of J.

If $h \geq 1$, then let T_{left} and T_{right} be the left and right subtrees of T. Thus T_{left} and T_{right} are complete ordered binary trees of height $h-1$. By induction, there exist complete binary subtrees $\widehat{T}_{left}$ and $\widehat{T}_{right}$ with the above properties computing the same functions as T_{left} and T_{right}, respectively. We build $\widehat{T}$ from $\widehat{T}_{left}$, $\widehat{T}_{right}$, and some special trees defined as follows. Let T_e ($e \geq 1$) be an ordered complete binary tree of height e whose internal nodes at even height are $\vee$-gates and whose internal nodes at odd height are $\wedge$-gates. Furthermore, all leaves of the left subtree of T_e are labeled with 0 and the leaves of the right subtree with 1. Note that T_e computes 1 iff e is even.

Now if the root of T is an $\wedge$-gate then define $\widehat{T} = (\widehat{T}_{left} \wedge \widehat{T}_{right}) \vee T_{2h+1}$ and if the root of T is an $\vee$-gate then define $\widehat{T} = (\widehat{T}_{left} \wedge T_{2h}) \vee (\widehat{T}_{right} \wedge T_{2h})$. Since, by induction, the trees $\widehat{T}_{left}$ and $\widehat{T}_{right}$ each have height $2h$, in both cases above $\widehat{T}$ is a complete binary tree of height $2h+2$. It is easily verified that $\widehat{T}$ computes the same function as T.

To complete the proof, we only need to show that the leaves of $\widehat{T}$ are labeled with a permutation of J. Since the height of $\widehat{T}$ is $2h+2$, it suffices to show that each of the variables $x_1, x_2, \ldots, x_n$ appears as a leaf of $\widehat{T}$ exactly twice, half of the remaining leaves are labeled with 0, and the other half labeled with 1. T_{left} (respectively, T_{right}) has leaves $x_1, x_2, \ldots, x_{n/2}$ ($x_{n/2+1}, \ldots, x_n$). Thus by induction, the variables $x_1, x_2, \ldots, x_{n/2}$ ($x_{n/2+1}, \ldots, x_n$) appear in $\widehat{T}_{left}$ ($\widehat{T}_{right}$) exactly twice, and the remaining leaves of $\widehat{T}_{left}$ ($\widehat{T}_{right}$) are split evenly between the constants 0 and 1. All leaves of the trees T_{2h} and T_{2h+1} are labeled with constants, exactly half with 0 and half with 1. We conclude that $\widehat{T}$ contains $2n$ leaves that are labeled with variables (two for each of the n variables) and half the remaining leaves are labeled with 0, and half with 1. $\square$

4 Properties of Prediction-Preserving Reductions

We first show that $\trianglelefteq$ is transitive, and then, as promised, we show that the reducibility is in fact prediction-preserving.

Lemma 4.1 *The relation $\trianglelefteq$ is transitive, i.e., if $R_1 \trianglelefteq R_2 \trianglelefteq R_3$, then $R_1 \trianglelefteq R_3$.*

Proof: Let R_1, R_2, and R_3 have implicit mappings c_1, c_2, c_3 (see Definition 2.2). Let $R_1 \trianglelefteq R_2$ be witnessed by functions f_a, g_a, t_a, and q_a, as described in Definition 3.1, and let $R_2 \trianglelefteq R_3$ be witnessed by the functions f_b, g_b, t_b, and q_b. Define f and g as follows:

$$f(w,s,n) = f_b(f_a(w,s,n), q_a(s,s,n), t_a(n,s,n))$$

and

$$g(r,s,n) = g_b(g_a(r,s,n), q_a(s,s,n), t_a(n,s,n)).$$

Now let $s, n \in \mathbb{N}$, $r \in R_1^{[s]}$, and $w \in \Sigma^{[n]}$. Then, by the properties of f_a and g_a,

$$w \in c_1(r) \Leftrightarrow f_a(w,s,n) \in c_2(g_a(r,s,n)). \tag{1}$$

Assuming without loss of generality that t_a and q_a are monotone nondecreasing in each argument, it follows that $f_a(w,s,n) \in \Sigma^{[t_a(n,s,n)]}$, and that $g_a(r,s,n) \in R_2^{[q_a(s,s,n)]}$. Thus, by the properties of f_b and g_b,

$$\begin{gathered} f_a(w,s,n) \in c_2(g_a(r,s,n)) \\ \Leftrightarrow \\ f_b(f_a(w,s,n), q_a(s,s,n), t_a(n,s,n)) \in c_3(g_b(g_a(r,s,n), q_a(s,s,n), t_a(n,s,n))). \end{gathered} \tag{2}$$

Combining (1) and (2), f and g satisfy the first requirement of Definition 3.1. It is easily verified that the other two requirements are satisfied as well. □

Lemma 4.2 *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 given in Definition 3.1. Without loss of generality, we assume that t and q are monotone non-decreasing in each parameter. We construct a polynomial-time prediction algorithm A for R. A takes its input parameters s, n, and ϵ, and passes the parameters $s' = q(s,s,n)$,

$n' = t(n, s, n)$, and ϵ to A'. Then for each example $(x, label)$, A computes $f(x, s, n)$ and passes the example $(f(x, s, n), label)$ to A'. Note that the examples A gives to A' have length at most n', and are examples of some concept whose representation $g(r, s, n) \in R'$ has length at most s'. A also gives the unlabeled word $f(w, s, n)$ to A', where w is the unlabeled word that A receives. To predict the label of w, A predicts exactly what A' predicts for the word $f(w, s, n)$.

For some polynomial p', A' predicts incorrectly with probability at most ϵ (with respect to the image under f of the original distribution) when given $p'(s', n', \frac{1}{\epsilon})$ examples. By (1) in the definition of a prediction-preserving reduction, the probability that A predicts incorrectly (with respect to the original distribution) is the same. Clearly $p'(s', n', \frac{1}{\epsilon})$ and the run time of A are polynomial in s, n, and $\frac{1}{\epsilon}$. $\square$

As discussed in Section 2, the definition of polynomial predictability does not necessarily require that the parameters s, n, and ϵ be given to the prediction algorithm. If they are not given, then in the proof of Lemma 4.2, the prediction algorithm A that is constructed for R by simulating a prediction algorithm A' for R' must first choose its parameters. As before, the parameters are chosen x, x, $\frac{1}{x}$, where x is the maximum integer such $\hat{p}(x) = p'(x, x, x)$ is at most as large as the number of examples. If x is undefined and if the length of the longest example exceeds x then A simply predicts $+$ by default. Otherwise the simulation proceeds as before using the chosen parameters and the first $\hat{p}(x)$ examples. It is easily shown that A predicts as required when given at least $\hat{p}(s' + n' + \frac{1}{\epsilon})$ examples.

Definition 4.3 *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 $\mathcal{R}$.*

Thus if a prediction problem R is prediction-hard for a class $\mathcal{R}$, then the predictability of R implies the predictability of every prediction problem in $\mathcal{R}$.

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

As we shall see, 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.) Consequently, all prediction problems whose evaluation problems have similar complexity form a natural class of prediction problems.

Definition 4.5 *For a class or languages $\mathcal{L}$, let $\mathcal{R}_{\mathcal{L}} = \{R : E(R) \in \mathcal{L}\}$.*

Below we consider collections of prediction problems $\mathcal{R}_{\mathcal{L}}$ where $\mathcal{L}$ is a complexity class.

5 Prediction-Complete Problems

For each of the standard complexity classes $\mathcal{NC}^1$, $\mathcal{LOG}, \mathcal{NLOG}, \mathcal{LOGCFL}$, and $\mathcal{P}$, we give a natural prediction problem that is prediction-complete for the corresponding classes of prediction problems $\mathcal{R}_{\mathcal{NC}^1}$, $\mathcal{R}_{\mathcal{LOG}}$, $\mathcal{R}_{\mathcal{NLOG}}$, $\mathcal{R}_{\mathcal{LOGCFL}}$, and $\mathcal{R}_{\mathcal{P}}$. In particular, we show that the problems of predicting Boolean formulas, DFAs, NFAs, PDAs, and alternating DFAs, are as hard as predicting any problem in the above classes of prediction problems, respectively. The main technique is given in detail for DFAs. The prediction-completeness of other automata prediction problems follow analogously, and the proofs will only be sketched.

Theorem 5.1 *R_{DFA} is prediction-complete for $\mathcal{R}_{\mathcal{LOG}}$.*

Theorem 5.1 is proved by showing that predictability of DFAs implies the predictability of logspace Turing machines. We need the following definitions. Let $\Delta = \Sigma \cup \Gamma \cup \{B\}$ (where B denotes the blank symbol). For each constant $k > 0$, we define

- $R_{k\ log\ TM} = \{r : r$ encodes a single tape (offline) TM with tape alphabet Δ that runs in space at most $k \log n$ on inputs of length $n\}$.

Lemma 5.2 *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 that accepts $c(r)$ and is of size $|T| + \mathcal{O}(|r|)$. (T_r is the TM T with r included in the state information.) Hence f and g witness that $R \trianglelefteq R_{k\ log\ TM}$, where for any s, r, n, and w, $f(w, s, n) = w$ and $g(r, s, n)$ is a representation of T_r. $\square$

Proof of Theorem 5.1: It is easy to show that $R_{DFA} \in \mathcal{R}_{\mathcal{LOG}}$, i.e., that it is logspace decidable whether a given DFA accepts a given word. To show that any $R \in \mathcal{R}_{\mathcal{LOG}}$ reduces to R_{DFA} we show that for each constant $k > 0$, $R_{k\ log\ TM} \trianglelefteq R_{DFA}$, and the result follows from Lemma 5.2 and Lemma 4.1.

Let r be the encoding of a TM $T \in R_{k\ log\ TM}$, and let s be an upper bound on $|r|$, and therefore, an upper bound on the size of the state set Q_T of T. Let the word transformation f be defined by $f(w,s,n) = 1^{|w|}0w^{p(|w|,s,n)}$, where p is a polynomial defined later. Let the representation transformation g be defined by $g(r,s,n) = M_T$, where M_T is described below.

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_T will, on input $f(w,s,n)$, simulate the action of T on input w.

M_T is composed of $n+1$ smaller DFAs, M_0, M_1, …, M_n. M_T uses a chain of $n+1$ states to read the initial string of 1's in $f(w,s,n)$, 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 $p(|w|,s,n)$ 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.[6]

$M_{|w|}$ 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. Since s is an upper bound on $|Q_T|$, this product is at most

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

and thus the number of states of M_T is at most

$$n+1+(n+1)s(n-1)(|\Delta|^{k \log n})(k \log n).$$

Thus the length of the encoding of M_T is at most polynomial in n and the length of the encoding of T. Further, the number $p(|w|,s,n)$ of copies of w that are required is at most the number of moves of T, which is at most the number of distinct configurations of T.

[6] In [51], two-way head movement is similarly reduced to one-way head movement by using a power of ww^R (where w^R is the reverse of w).

Since T is space bounded by $k \log n$, the number of distinct configurations is polynomial in n and $|Q_T| \leq s$. $\square$

It should be clear that the proof of Theorem 5.1 may be applied to any reasonable class of automata. The theorem essentially says that, with respect to predictability, there is no distinction between an automaton and the two-way version of the automaton with an additional logspace work tape. This observation provides us with the next three corollaries.

Corollary 5.3 *R_{NFA} is prediction-complete for $\mathcal{R}_{\mathcal{NLOG}}$.*

Proof: To show prediction-hardness, repeat the proof of Theorem 5.1 using the corresponding nondeterministic version of DFAs and TMs. It is easily verified the $E(R_{NFA})$, the evaluation problem for NFA membership, is in $\mathcal{NLOG}$. $\square$

For each polynomial h, let

- $R_{hPDA} = \{r : r \text{ encodes a } h(n) \text{ time bounded } PDA\}$.

Corollary 5.4 *R_{CFG} is prediction-complete for $\mathcal{R}_{\mathcal{LOGCFL}}$, and there exists a polynomial h such that R_{hPDA} is prediction-complete for $\mathcal{R}_{\mathcal{LOGCFL}}$.*

Proof: Recall that $\mathcal{LOGCFL}$ is the class of languages accepted by a polynomially time bounded two-way PDA with auxiliary logspace work tape [52]. To see that R_{PDA} is prediction-hard for $\mathcal{R}_{\mathcal{LOGCFL}}$, apply the proof of Theorem 5.1 with PDAs replacing DFAs, and with polynomially time bounded two-way PDAs with additional logspace replacing logspace TMs.

To see that R_{CFG} is also prediction-hard for $\mathcal{R}_{\mathcal{LOGCFL}}$, observe that for any PDA M there is a CFG G in Chomsky normal form[7] of size polynomial in the size of M, that generates the same language [25]. Thus $R_{PDA} \trianglelefteq R_{CFG}$ (the representation transformation maps a PDA to the equivalent CFG, and the word transformation is the identity on w), and it follows that R_{CFG} is also prediction-hard for $\mathcal{R}_{\mathcal{LOGCFL}}$. Prediction-completeness for R_{CFG} follows by showing $E(R_{CFG}) \in \mathcal{LOGCFL}$, which is straightforward from the definition of $\mathcal{LOGCFL}$.

To prove the second 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 h and h' such that for each grammar G there is a PDA M_G of size at

[7]We use the definition in [25], which allows the production $S \to \lambda$.

most $h'(|G|)$ such that $\{w : M_G \text{ accepts } w \text{ in at most } h(|w|) \text{ steps}\} = L(G)$. It follows that $R_{CFG} \trianglelefteq R_{hPDA}$ (with word transformation f as the identity on its first argument, and representation transformation g that maps the representation of a grammar G to the representation of the PDA M_G), and thus R_{hPDA} is prediction-hard for $\mathcal{R}_{\mathcal{LOGCFL}}$.

To see that $E(R_{hPDA}) \in \mathcal{LOGCFL}$, note that there is a universal 2-way auxiliary PDA U that on input of an encoding r of any $h(n)$ time bounded PDA M and string w, U simulates M on input w: U uses at most $O(\log(|r| + |w|))$ space on its auxiliary worktape to store the current location of M's read head and the current state of M while scanning across the input representation r looking for an applicable transition. For some polynomial p independent of M, U can simulate M with at most $p(|r| + |w|)$ steps for each step of M. Thus the total time taken by U is at most $p(|r| + |w|)h(|w|)$ which is a polynomial in the size of U's input. Since U is a 2-way PDA running in polynomial time and using at most logspace auxiliary worktape, we conclude that $E(R_{hPDA}) \in \mathcal{LOGCFL}$. □

Recall that an alternating DFA is a DFA where transitions may be of three types: existential (nondeterministic), universal, or negation. (For a formal definition, see [15].) Defining the prediction problem R_{altDFA} in the obvious way, we prove the following corollary.

Corollary 5.5 *R_{altDFA} is prediction-complete for $\mathcal{R}_{\mathcal{P}}$.*

Proof: An alternate characterization of $\mathcal{P}$ is given in terms of alternating logspace TMs [15]. In particular, $\mathcal{P}$ = alternating logspace. Note that an alternating logspace TM is simply the two-way version of an alternating DFA, with additional logspace work tape. To see that R_{altDFA} is prediction-hard for $\mathcal{R}_{\mathcal{P}}$, apply the proof of Theorem 5.1 with alternating DFAs replacing DFAs, and with alternating logspace TMs replacing standard logspace TMs. To show prediction-completeness, we must show $E(R_{altDFA}) \in \mathcal{P}$. There is an alternating TM that, on input the description of an alternating DFA M and input word w, simulates M on input w and accepts iff M accepts w. Clearly only logspace is needed for such a simulation, thus $E(R_{altDFA}) \in$ alternating logspace $= \mathcal{P}$. □

Theorem 5.6 *R_{BF} is prediction-complete for $\mathcal{R}_{\mathcal{NC}^1}$.*

Proof: Let R be any prediction problem such that $E(R) \in \mathcal{NC}^1$. Thus there is an alternating logtime Turing machine A accepting $E(R)$. By "hardwiring" the representation into the state space of the machine we have that for all $r \in R$ there is an alternating logtime Turing machine A_r that accepts $c(r)$. Furthermore for each length bound n there is a modified alternating logtime Turing machine $A_{r,n}$ that accepts all words of $c(r)$ of length at most n, padded to have equal length $n+1$. More precisely, $A_{r,n}$ accepts the language $\{w01^{n-|w|} : |w| \leq n \text{ and } w \in c(r)\}$. The machine $A_{r,n}$ can be chosen so that its size is polynomial in $|r|$ and n. In more detail, let $|A_{r,n}|$ denote the number of symbols required to encode the machine with respect to some standard encoding scheme. Then for some polynomial q (which may depend on R), for all $r \in R$ and $n \in \mathbb{N}$, $|A_{r,n}| \leq q(|r|, n)$. By Theorem 2 of [14] there exists a polynomial p such that for any such $A_{r,n}$ there exists a Boolean formula $F_{r,n}$ of size at most $p(n, |A_{r,n}|)$ accepting the same language as $A_{r,n}$.

Now the following transformations witness the fact that $R \trianglelefteq R_{BF}$: Let the word transformation $f(w, s, n) = w01^{n-|w|}$, and the representation transformation $g(r, s, n)$ be the representation in R_{BF} for $F_{r,n}$. To complete the proof, we observe that $E(R_{BF})$ has been shown to be in alternating logtime by Buss [14] (see Main Theorem 1 and the comments on top of page 129). Thus $E(R_{BF}) \in \mathcal{NC}^1$. □

By Theorems 3.3 and 3.4, R_{TREE_1} and R_{TREE_2} are prediction-hard for $\mathcal{R}_{\mathcal{NC}^1}$. It is also easy to show that these problems are prediction-complete for $\mathcal{R}_{\mathcal{NC}^1}$.

We use the above theorem to show that $R_{BF} \trianglelefteq R_{DFA}$.

Corollary 5.7 *R_{DFA} is prediction-hard for $\mathcal{R}_{\mathcal{NC}^1}$ and in particular, for any of the representations $R \in \{R_{BF},\ R_{TREE_1},\ R_{TREE_2}\}$, $R \trianglelefteq R_{DFA}$.*

Proof: By Theorem 5.1 R_{DFA} is prediction-complete for $\mathcal{R}_{\mathcal{LOG}}$. Thus $R_{BF} \trianglelefteq R_{DFA}$ follows from the fact that $\mathcal{R}_{\mathcal{NC}^1} \subseteq \mathcal{R}_{\mathcal{LOG}}$, which is true because $\mathcal{NC}^1$ = alternating logtime $\subseteq \mathcal{LOG}$ [15, 58]. The other two problems now reduce to R_{DFA} by Theorems 3.3 and 3.4. □

An alternate reduction from Boolean formula predictability to DFA predictability may be given as follows: $R_{BF} \trianglelefteq R_{TREE_1}$ (Theorem 3.3), and the evaluation problem for R_{TREE_1} is trivially in $\mathcal{LOG}$. $R_{BF} \trianglelefteq R_{DFA}$ now follows from Theorem 5.1.

In Section 7 we discuss recent results [39] suggesting that neither R_{BF} nor R_{DFA} are likely to be predictable.

A natural open problem that is captured by R_{CONVEX} is that of predicting membership in an unknown convex polytope. Recall from Section 3 that $R_{CNF} \trianglelefteq R_{CONVEX}$. We now show that $R_{CONVEX} \trianglelefteq R_{BF}$. In Section 6 we consider a generalization of this problem, and show that predicting whether cubes have nonempty intersection with an unknown convex polytope is prediction-complete for $\mathcal{R}_{\mathcal{P}}$, and is thus unlikely to be predictable by results in Section 7.

Corollary 5.8 *$E(R_{CONVEX}) \in \mathcal{NC}^1$, and thus for any $R \in \{R_{BF},\ R_{TREE_1},\ R_{TREE_2}\}$, $R_{CONVEX} \trianglelefteq R$.*

Proof: To show that $E(R_{CONVEX}) \in \mathcal{NC}^1$, we show that it can be determined in alternating logtime whether a given input vector $\vec{x}$ satisfies all of the inequalities $\vec{a_i} \cdot \vec{x} \geq b_i$, where the vectors $\{\vec{a_i}\}$ and the numbers $\{b_i\}$ are also given as input. To do this, it is sufficient to show that it can be determined in alternating logtime whether a single inequality $\vec{a_i} \cdot \vec{x} \geq b_i$ holds, since a single universal branch of an alternating TM could check in parallel the input inequality for each i.

We will say that a function f can be "computed" in alternating logtime if the associated language L_f is in alternating logtime, where $L_f = \{\langle z, i, c\rangle$: the i-th character of $f(z)$ is $c\}$. Standard techniques [49] can be adapted to show that the sum of n integers, each of n bits, and that the multiplication of two n-bit integers, can be computed in alternating logtime. This implies that the dot-product $\vec{a} \cdot \vec{x}$ can be computed in alternating logtime. I.e., the language $L_{dot} = \{\langle \vec{a}, \vec{x}, i, c\rangle$: the i-th bit of $\vec{a} \cdot \vec{x}$ is $c\}$ is in alternating logtime.

Let $z = \vec{a} \cdot \vec{x}$. For any number u, let $u[i]$ be the i-th bit from the right in the binary representation of u ($u[0]$ is the least significant bit of u). To determine, in alternating logtime, whether $z \geq b$, an existential branch guesses the most significant bit i such that $z[i] \neq b[i]$. It is then determined via a universal branch, each branch using the alternating logtime membership test for L_{dot}, that for all sufficiently large $j > i$, $z[j] = b[j]$, and that $z[i] = 1 > 0 = b[i]$. □

We conclude this section by discussing some differences between our notion of prediction-preserving 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 [34], $E(R_{DFA})$ (the membership problem for DFAs) is complete for $\mathcal{LOG}$ with respect to one-way, read-only-input, deterministic logspace reductions. Similarly, R_{NFA} is prediction-complete for $\mathcal{R}_{\mathcal{NLOG}}$, and $E(R_{NFA})$ is complete for $\mathcal{NLOG}$ 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 empty concept $\{\}$. Now $E(R)$ is complete for $\mathcal{L}$, but R is trivially predictable.

6 Prediction Problems Complete for $\mathcal{P}$

In the previous section we saw that alternating DFAs are prediction-complete for $\mathcal{R}_{\mathcal{P}}$. In this section we give a number of additional prediction problems with this property.

From any language L that is not in $\mathcal{P}$, it is easy to construct a prediction problem R_L that is trivially predictable, even though $E(R_L)$ is not in $\mathcal{P}$: Let R_L consist of all words w over the same alphabet as L, where $c(w) = \{w\} \cap L$. Thus, the concept represented by a word w is either the empty language, or the language consisting of the singleton w, depending on whether $w \in L$. Clearly, $E(R_L)$ has the same complexity as L, and is trivially predictable.

Even though it is easy to construct prediction problems outside $\mathcal{R}_{\mathcal{P}}$, most "natural" problems lie in this class. Consequently, prediction problems that are prediction-complete for $\mathcal{R}_{\mathcal{P}}$ are of particular interest, as their predictability would imply the predictability of any prediction problem in which we might be reasonably interested. In Section 7, we give strong evidence that these problems are not predictable.

We first show that prediction of Boolean circuits is prediction-complete for $\mathcal{R}_{\mathcal{P}}$.

- $R_{CIRC} = \{r : r \text{ encodes an acyclic Boolean circuit}\}$, 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.

Theorem 6.1 *R_{CIRC} is prediction-complete for $\mathcal{R}_{\mathcal{P}}$.*

Proof: Clearly $E(R_{CIRC}) \in \mathcal{P}$. 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 n there is a Boolean circuit $B_{r,n}$ that accepts all words of $c(r)$ of length at most n, padded so as to have equal length. More precisely, let $B_{r,n}$ accept the language $\{w01^{n-|w|} : |w| \leq n$ and $w \in c(r)\}$. Thus $B_{r,n}$ has $n+1$ inputs. Since $E(R) \in \mathcal{P}$, there exists a polynomial q such that for all s, $r \in R^{[s]}$, and $n \in \mathbb{N}$, $B_{r,n}$ can be chosen such that $|B_{r,n}| \leq q(|r|, s, n)$. Now let the word transformation $f(w, s, n) = w01^{n-|w|}$, and the representation transformation $g(r, s, n) = B_{r,n}$. $\square$

Our goal now is to reduce R_{CIRC} to other natural prediction problems (which have polynomial time associated evaluation problems), thus showing that they are prediction-complete for $\mathcal{R}_\mathcal{P}$. 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 exactly the *Circuit Value Problem* (CVP), the standard problem that is complete for $\mathcal{P}$ with respect to logspace reductions [35, 40]. CVP has been reduced to a large number of other problems in $\mathcal{P}$, thus showing that these problems are also $\mathcal{P}$-complete [33]. In many cases a logspace reduction from CVP 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. The prediction problems defined below, and the proofs that they are prediction-complete for $\mathcal{R}_\mathcal{P}$, were obtained in this manner.

The first two prediction problems are defined with respect to the same set of representations. Recall that a Horn clause is a disjunction of literals, where at most one literal is unnegated. Equivalently, a Horn clause is either an implication $y \Leftarrow x_1 \wedge x_2 \wedge \cdots \wedge x_n$, or the assertion $y \Leftarrow$ TRUE, where y and each x_i is an unnegated literal. Let R_{HC} be the set of representations $\{r : r$ 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 (assertions) that are consistent with the conjunction represented by r

An almost identical prediction problem, that was proved prediction-complete for $\mathcal{R}_\mathcal{P}$ by Angluin [4], is:

- 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.

Additional prediction problems that we consider are:

- Augmented CFG emptiness: $R_{\emptyset CFG} = \{r : r$ encodes a CFG $\}$, where $c(r)$ is the set of all collections of productions that when added to the grammar represented by r, yield a context free grammar that generates the empty language.
- Convex polytope intersection: $R_{POLYTOPE} = \{r : r$ 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 $l\}$, where $c(r)$ is the set of unit subcubes that have non-empty intersection with the polytope represented by r.

Theorem 6.2 *Let $R \in \{(R_{HC}, c), (R_{HC}, c'), R_{\emptyset CFG}, R_{POLYTOPE}\}$. Then $R_{CIRC} \trianglelefteq R$, and thus each of these problems is prediction-complete for $\mathcal{R}_\mathcal{P}$.*

Each of the prediction problems in Theorem 6.2 has an evaluation problem decidable in polynomial time, thus it is sufficient to prove prediction-hardness for $\mathcal{R}_{\mathcal{P}}$. Each of the following prediction-preserving reductions correspond to a proof of $\mathcal{P}$-completeness of a related decision problem. The proofs of $\mathcal{P}$-completeness are all achieved via logspace reductions from CVP, i.e., given a truth assignment and an acyclic circuit, does the last gate of the circuit evaluate to 1? We repeat the known logspace reductions from CVP within the framework of prediction-preserving reductions.

Proof that $R_{CIRC} \trianglelefteq (R_{HC}, c)$ **:**
Horn clause consistency is related to the $\mathcal{P}$-complete problem of deciding whether the empty clause can be deduced from a given set of Horn clauses [35]. We use the method of [33] to show that R_{CIRC} reduces to the prediction problem of Horn clause consistency.

Let $r \in R_{CIRC}$ be the encoding of an acyclic circuit U_r with n inputs $x_1, \ldots, x_n$, and gates $x_{n+1}, \ldots, x_l$. Without loss of generality, all gates are $\wedge$-gates or $\neg$-gates. We reduce R_{CIRC} to the Horn clause consistency problem using transformations f and g as defined below.

Let $g(r, s, n) = h$, where h represents a conjunction of Horn clauses obtained as follows. The variables of h are $X_1, \ldots, X_l$ and $Y_1, \ldots, Y_l$ (each Y_i will represent the negation of X_i). For each gate $x_i = \neg x_j$, include in the conjunction h the clauses $X_i \Leftarrow Y_j$ and $Y_i \Leftarrow X_j$. For each gate $x_i = x_j \wedge x_k$, include in the conjunction h the clauses $X_i \Leftarrow X_j \wedge X_k$, $Y_i \Leftarrow Y_j$, and $Y_i \Leftarrow Y_k$.

If $w = w_1 w_2 \cdots w_n$ is an n bit string input to the circuit, and s is any upper bound on the length of the unknown circuit, then let $f(w, s, n)$ be the collection of facts (assertions) $\{B_i \Leftarrow \text{TRUE} : 1 \leq i \leq n\} \cup \{X_l \Leftarrow \text{TRUE}\}$, where $B_i = X_i$ if $w_i = 1$, and $B_i = Y_i$ if $w_i = 0$.

It is easily verified that U_r accepts the input w (i.e., gate x_l evaluates to 1) if and only if the collection of assertions $f(w, s, n)$ is consistent with the conjunction of clauses $h = g(r, s, n)$. Further, f is polynomial-time computable, and g is polynomial length preserving. $\square$

Proof that $R_{CIRC} \trianglelefteq (R_{HC}, c')$ **:**
Prediction of Horn clause implication was shown to be prediction-complete for $\mathcal{R}_{\mathcal{P}}$ by Angluin [4], using the methodology of prediction-preserving reductions introduced here. Her proof is nearly identical to that for Horn clause consistency, given above: The representation transformation $g(r, s, n)$ is identical. The word transformation $f(w, s, n)$ encodes the clause $X_l \Leftarrow B_1 \wedge B_2 \wedge \cdots \wedge B_n$, where $B_i = X_i$ if $w_i = 1$, and $B_i = Y_i$ if $w_i = 0$. It is easily verified that U_r accepts w if and only if the collection of clauses $g(r, s, n)$ implies the clause $f(w, s, n)$. $\square$

Proof that $R_{CIRC} \trianglelefteq R_{\emptyset CFG}$:
The proof of the prediction-completeness of Augmented CFG emptiness ($R_{\emptyset CFG}$) is related to the proof of $\mathcal{P}$-completeness of deciding for a given CFG G, whether $L(G) = \emptyset$ [24, 35]. The reduction relies on the reduction above for Horn clause consistency. In the reduction, the Horn clauses become the productions of the CFG.

Again, let $r \in R_{CIRC}$ be the encoding of an acyclic circuit U_r with n inputs $x_1, \ldots, x_n$, and gates $x_{n+1}, \ldots, x_l$, all either $\wedge$-gates or $\neg$-gates without loss of generality. Let $g(r, s, n)$ be the grammar S, N, T, P as follows. The start symbol S is the symbol Y_l. The set of nonterminals $N = \{X_1, \ldots, X_l, Y_1, \ldots, Y_l\}$. The set of terminal symbols T is empty, thus the grammar will generate either the empty language, or the language consisting only of the empty string λ. The set of productions P is constructed as follows. For each gate $x_i = \neg x_j$ of U_r, include in P the productions $X_i \rightarrow Y_j$ and $Y_i \rightarrow X_j$. For each gate $x_i = x_j \wedge x_k$, include in P the productions $X_i \rightarrow X_j X_k$, $Y_i \rightarrow Y_j$, and $Y_i \rightarrow Y_k$. In this construction, a derivation corresponds to starting at the output gate (start symbol) Y_l, and "computing backwards".

The word transformation $f(w, s, n)$ now must provide an additional collection of productions, such that when added to the grammar $g(r, s, n)$, the resulting language is empty iff the circuit U_r accepts $w = w_1 w_2 \cdots w_n$. Define $f(w, s, n)$ to be the productions $\{B_i \rightarrow \lambda : 1 \leq i \leq n\}$, where $B_i = X_i$ if $w_i = 1$, and $B_i = Y_i$ if $w_i = 0$.

In the $\mathcal{P}$-completeness proof of [24], it is shown that the last gate x_l of U_r computes 1 on input w iff the CFG with productions $f(w, s, n) \cup g(r, s, n)$ and start symbol Y_l derives the empty language. $\square$

Proof that $R_{CIRC} \trianglelefteq R_{POLYTOPE}$:
The convex polytope intersection prediction problem, $R_{POLYTOPE}$, is related to deciding whether there exists any feasible solution to a given linear programming problem. This problem is $\mathcal{P}$-complete [17]. Again, we adapt the reduction from CVP used to show that linear programming feasibility is $\mathcal{P}$-complete.

Let $r \in R_{CIRC}$ be the encoding of an acyclic circuit U_r with n inputs $x_1, \ldots, x_n$, and gates $x_{n+1}, \ldots, x_l$, all either $\wedge$-gates or $\neg$-gates without loss of generality. Let $g(r, s, n)$ be the linear programming problem over variables $x_1, \ldots, x_l$, consisting of inequalities that encode the gates of U_r. The inequalities $g(r, s, n)$ define a convex polytope contained in the l-dimensional unit cube. For each gate $x_i = \neg x_j$, we include the inequalities of $x_i = 1 - x_j$; for each gate $x_i = x_j \wedge x_k$ we include the inequalities $0 \leq x_i \leq 1$, $x_i \leq x_j$, $x_i \leq x_k$, and $x_j + x_k - 1 \leq x_i$. Further, for the output gate x_l of r, include the inequalities of $x_l = 1$.

The word transformation $f(w, s, n)$ consists of inequalities corresponding to an $l - n$ dimensional subcube of the l dimensional unit cube given. In particular, $f(w, s, n)$ includes the inequalities of: $x_i = w_i$, for $1 \leq i \leq n$, and $0 \leq x_j \leq 1$, for $n + 1 \leq j \leq l$.

Reinterpreting the reduction of [17], the gate x_l of the circuit U_r outputs 1 iff the collection of inequalities $g(r, s, n) \cup f(w, s, n)$ contains a feasible solution, i.e., iff the subcube defined by $f(w, s, n)$ has nonempty intersection with the polytope defined by $g(r, s, n)$. $\square$

An additional prediction problem that we mention informally is that of predicting *Fixed Grid Planar Circuits*. It is easy to show that any Boolean circuit can be embedded into an at most polynomially larger square grid, using $\wedge$, $\vee$, $\neg$, and selection gates at the nodes. It follows that this restricted class of circuits is also prediction-complete for $\mathcal{R}_{\mathcal{P}}$. Thus, as shown in Section 3 for Boolean trees (Theorem 3.3), restricting the circuit topology in a uniform way does not result in an easier prediction problem.

Lately, considerable attention has been given to neural networks as models of computation, particularly in the context of learning. In one of the basic models, a network is a circuit, where each gate computes a linear function of its inputs, and evaluates to 1 iff the value exceeds a given threshold. It is easy to show that $\wedge$, $\vee$, $\neg$, and selection gates can be simulated with threshold elements using only small integral weights. Thus the prediction problem for fixed grid planar neural networks is also prediction-complete for $\mathcal{R}_{\mathcal{P}}$.

Adleman [1] showed that there are non-uniform polynomially sized circuit families for all languages in $\mathcal{RP}$, the class of languages accepted in randomized polynomial time [20]. More precisely, for each language $L \in \mathcal{RP}$, there is a polynomial p and a (non-uniform) family of circuits $\{C_n\}_{n\geq 1}$ such that C_n accepts $L \cap \{0,1\}^n$ and such that C_n has size at most $p(n)$. By a straightforward adaptation of the proof of Theorem 6.1, this implies that R_{CIRC} is prediction-complete for $\mathcal{R}_{\mathcal{RP}}$. This follows from the fact that the representation transformation in a prediction-preserving reduction is highly non-uniform — it need not even be computable. By transitivity, all other prediction-complete problems for $\mathcal{R}_{\mathcal{P}}$ are also prediction-complete for $\mathcal{R}_{\mathcal{RP}}$.

7 Unpredictable Concepts

We give evidence that the prediction problems complete for $\mathcal{R}_{\mathcal{P}}$ discussed in Section 6 are not predictable, even in a very weak sense. We also discuss recent results [39], showing that Boolean formulas and DFAs are not (even weakly) predictable, relying on certain cryptographic assumptions.

Definition 7.1 *The prediction problem R is* weakly (polynomially) predictable *iff there* *exists a polynomial-time prediction algorithm A and polynomials p and q such that for all input parameters s and n, for all $r \in R^{[s]}$, and for all probability distributions on $\Sigma^{[n]}$, if A is given at least $p(s,n)$ randomly generated examples of (the unknown target concept) $c(r)$, and a randomly generated unlabeled word $w \in \Sigma^{[n]}$, then the probability that A incorrectly predicts $label_{c(r)}(w)$ is at most $\frac{1}{2} - \frac{1}{q(s,n)}$.*

The difference between predictability and weak predictability is that to satisfy the definition of weak prediction, an algorithm need only be able to predict with accuracy exceeding $\frac{1}{2}$ by a vanishing (as s and n grow) fraction of the distribution, whereas for predictability, any accuracy arbitrarily close to 1 must be achievable. Clearly predictability implies weak predictability. Surprisingly, Schapire [50] proves the converse, giving the following theorem:

Theorem 7.2 [50] *For all prediction problems R, R is polynomially predictable if and only if R is weakly polynomially predictable.*

Our next lemma asserts that prediction-preserving reducibility also preserves weak predictability.

Lemma 7.3 *For all prediction problems R and R', if $R \trianglelefteq R'$ and R' is weakly predictable, then R is weakly predictable.*

Proof: The proof is a trivial variant of the proof of Lemma 4.2, using weak predictability instead of predictability. Alternatively, if R' is weakly predictable, Theorem 7.2 asserts that R' is predictable, and Lemma 4.2 shows that R is predictable, and thus weakly predictable. $\square$

Assuming the existence of a cryptographically secure pseudorandom bit generator (CSB generator) Goldreich, Goldwasser, and Micali [22] have constructed a collection of polynomial-time computable functions that are indistinguishable (to any polynomially time bounded algorithm) from functions "chosen at random".

We rephrase the results of [22] in our terminology. Based on the CSB generator, a polynomial-time algorithm (without loss of generality, a TM) T is constructed. T expects two inputs of the same length. Call the first input the index. Each index $r \in \Sigma^n$ defines a new TM T_r such that $T_r(w)$ accepts iff $T(r,w)$ accepts. Note that $|T_r|$ is $\mathcal{O}(|r|)$. Let $R_{HARD(n)}$ = the set of encodings of machines T_r for each $r \in \Sigma^n$. Let $R_{HARD} = \cup_n R_{HARD(n)}$. Since each T_r is essentially the same polynomial-time Turing machine T, $E(R_{HARD}) \in \mathcal{P}$.

Goldreich *et al* prove that R_{HARD} has the following property. Let A be any polynomial-time *query-test* algorithm for R_{HARD}, where a query-test algorithm A works as follows: A, given n as input, attempts to predict those words of length n accepted by some randomly chosen element $r \in R_{HARD(n)}$ by first *querying* on any collection of words of length n and of cardinality polynomial in n, as to whether or not the machine encoded by r accepts. Then A chooses a different test word w (of length n) and predicts whether w is accepted by the machine encoded by r. Let $avg\text{-}error_n(A)$ denote the probability that A is incorrect, where the probability is taken over the random selection of $r \in R_{HARD(n)}$ 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, $avg\text{-}error_n(A) > \frac{1}{2} - \frac{1}{p(n)}$, for sufficiently large n.

Lemma 7.4 *Assuming that CSB generators exist, R_{HARD} is not weakly predictable.*

Proof: Suppose to the contrary that there exists a prediction algorithm B that weakly predicts R_{HARD}. Let A be a query-test algorithm that runs B using a uniform distribution on words of length n. To supply an example to B, A simply generates a word of length n randomly, and queries whether it is a positive or negative example. By the definition of weakly predictable, after only polynomially many examples (and time), regardless of the unknown $r \in R_{HARD(n)}$, B will be able to correctly predict $label_{c(r)}(w)$ for a randomly generated w of length n, with probability at least $\frac{1}{2} + \frac{1}{q(|r|,n)} = \frac{1}{2} + \frac{1}{q(n,n)}$ for some polynomial q. A supplies sufficiently many examples for this guarantee to hold, and then randomly generates w as a test string, predicting $label_{c(r)}(w)$ as predicted by B.

Let $p(n) = q(n,n)$. Then for each $r \in R_{HARD(n)}$, the error of the query-test algorithm A is at most $\frac{1}{2} - \frac{1}{p(n)}$. Thus the average error satisfies $avg\text{-}error_n(A) \leq \frac{1}{2} - \frac{1}{p(n)}$, contradicting the fact above that no such query-test algorithm A for R_{HARD} exists with these properties. $\square$

Thus assuming that CSB generators exist, there is a prediction problem in $\mathcal{R}_\mathcal{P}$ that is not weakly predictable, and certainly not predictable. Similarly, it has been shown assuming the existence of CSB generators, that polynomial size Boolean circuits are not pac-learnable [10].

Theorem 7.5 *If there exists a CSB generator, then any prediction problem that is prediction-hard for $\mathcal{R}_\mathcal{P}$ is not weakly predictable.*

Proof: Follows immediately from Lemma 7.4. $\square$

Corollary 7.6 *Assuming the existence of any CSB-generators, R_{CIRC}, $R_{\emptyset CFG}$, $R_{POLYTOPE}$, (R_{HC}, c), and (R_{HC}, c'), are not weakly predictable.*

The existence of CSB-generators follows from the existence of one-way permutations [57], and is equivalent to the existence of one-way functions that are *one-way on iterates* [23, 41]. Consequently, the above theorems hold under the assumption that such one-way functions exist.

At this point a strategy for proving unpredictability based on cryptographic results becomes evident. If cryptographic functions that are hard to invert can be shown to have easy evaluation problems, (i.e., are computable in polynomial time, logspace, or lower in the complexity hierarchy) then any prediction problem that is prediction-hard for the relevant complexity class will not be predictable.

Kearns and Valiant [36, 39] have very recently taken this approach, and have shown that, based on certain specific cryptographic assumptions (the intractability of inverting the RSA cryptosystem, factoring Blum integers, or deciding quadratic residues), there are some prediction problems that are not weakly predictable. It is further shown that for each representation r and input length n there is a circuit $C_{r,n}$ that accepts $c(r) \cap \{0,1\}^n$, and such that $C_{r,n}$ has depth $O(\log(|r|+n))$ and size polynomial in $|r|$ and n. They observe that such circuits can be unfolded into equivalent Boolean formulas of size polynomial in $|r|$ and n, and they conclude that R_{BF} is not weakly predicatable based on the same cryptographic assumptions. By combining this result with Corollary 5.7 ($R_{BF} \trianglelefteq R_{DFA}$), they show that the same holds for DFA prediction.

8 Conclusion

Summary of results

We have developed the beginnings of a complexity theory for predictability. By considering a type of prediction-preserving reduction, we have been able to relate the difficulty of various prediction problems. To illustrate the usefulness of this tool, a number of example reductions were given. In particular, it was shown (Theorems 3.3 and 3.4) that even when given significant additional information about the structure of an unknown Boolean formula, the prediction problem is no easier then the prediction of unrestricted Boolean formulas.

This work was motivated in an attempt to understand the difficulty of learning DFAs from examples. We showed that DFAs are at least as hard to predict as any language in $\mathcal{LOG}$ (Theorem 5.1). Consequently, the prediction of DFAs is at least as hard as the prediction of Boolean formulas (Corollary 5.7) and convex regions (Corollary 5.8).

The approach of showing a prediction problem complete for a particular class of problems gave rise to an interesting parallelism between prediction of automata and standard complexity classes. For each of a number of classes, it was shown that there is a prediction problem whose predictability implies the predictability of all prediction problems whose associated evaluation problem lies in the complexity class (Theorem 5.1, Corollaries 5.3, 5.4, 5.5, Theorem 5.6).

Of particular interest are those prediction problems that are prediction-complete for polynomial time (Theorems 6.1 and 6.2). Besides implying the predictability of all reasonable prediction problems, assuming the existence of a type of one-way function, these prediction problems are not predictable even in an extremely weak sense.

Further, relying on specific cryptographic schemes, the results of [39] show that this predictive difficulty actually occurs at a significantly lower level of the complexity hierarchy. Thus it is no surprise that general algorithms have not been found for predicting DFAs or Boolean formulas.

It is interesting to note that all of the standard complexity classes we have discussed consist of languages that are efficiently decidable, where "efficient" ranges from polynomial time down to very fast parallel circuits. However, in the case of predictability, the corresponding classes of prediction problems from $\mathcal{R}_{\mathcal{P}}$ down to the "easiest" class $\mathcal{R}_{\mathcal{NC}^1}$ consist of prediction problems that are evidently very difficult. Thus we have provided a taxonomy of unpredictable problems, as opposed to predictable ones. In some sense, the largest complexity class of prediction problems that have known polynomial-time prediction algorithms is the class of constant depth 1 (unbounded fan-in) Boolean circuits (monomials) [53]. The problem of predicting constant depth 2 circuits is exactly the problem of predicting DNF or CNF formulas, which remains a basic open question in this field.

Relaxations of our model

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, the most natural (and general) extension would be to allow a reduction similar to a randomized Turing reduction, as opposed to the many-one reduction presented here. This would allow the use of a prediction algorithm for one concept class as an oracle for solving another prediction problem. We are unaware of any examples for which this more general reduction is required to prove reducibility from one problem to another in the context of learning from examples only.

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, 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. However, 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 in this paper. See [55] for more general definitions along these lines.

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 of an unknown concept as would be required to determine membership in the concept when given as input an example word and the representation of the concept. However, for all of the representations we have considered, concept membership could be evaluated in polynomial time; consequently, this additional allowance would be superfluous.

All of our reductions and proofs of completeness are not sensitive to the particular definition of distribution-independent predictability we have chosen. For example, they immediately held in the case of weak predictability discussed in Section 7.

A non-probabilistic model of polynomial-time predictability could be defined based on the "on-line" prediction model appearing in [9, 43]. In that model, the prediction algorithm is given an unbounded sequence of (unlabeled) 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 incorrect predictions (in the worst case) for any representation $r \in R^{[s]}$ and sequence of words from $\Sigma^{[n]}$, expressed as a function of s and n.

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 n. The transformations discussed in [5, 43] imply that if a prediction problem is polynomially predictable in this mistake-bounded model, then it is also polynomially predictable according to the Definition 2.6. The theorems of this paper concerning polynomial reducibility and completeness also hold for the mistake bounded model, and, in particular, the negative results proved here hold for the mistake bounded model as well. Note that 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).

Duality

Following Assouad [8], one can associate a *dual* prediction problem with each prediction problem as follows. In a (*primal*) prediction problem the representation of the target concept is unknown and labeled examples of the target concept are given. In the dual problem, positive examples consist of representations whose induced concepts all contain some unknown example. (E.g., in the dual DFA prediction problem the positive examples of the word 011 are representations of DFAs that accept the unknown word 011.)

The above definition can be made formal; the dual of the dual is the primal problem, and the evaluation problems for the primal and the dual are identical. It is interesting to note that for all prediction-complete problems given in this paper, the dual prediction problem is also prediction-complete for the same class as the primal prediction problem, even though this does not seem to be true in general.

While duality of prediction problems is of mathematical interest, the duals of most prediction problems are not sufficiently natural to be of interest. Perhaps the notion of duality will be useful in determining the complexity of prediction problems.

Open problems

The apparent difficulty of DFA prediction can be extended to other problems. Ehrenfeucht and Haussler [18] give a (not quite polynomial time) $n^{\log n}$ algorithm for learning (and hence predicting) decision trees. A natural open problem suggested by Haussler [28] 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 [26]. However, strictly ordered decision graphs are essentially "unrolled" DFAs, thus these decision graphs are at least as hard to predict as DFAs.

There are a large number of open problems. For particular prediction problems, it would be of interest to determine the relative difficulty of predictability. For example, can it be proven that DNF prediction is easier than prediction of all Boolean formulas? What is the largest class of prediction problems for which R_{DNF} is prediction-hard? Decision tree prediction trivially reduces to DNF prediction; is there a reduction in the other direction? Is Boolean formula prediction equivalent to DFA prediction? Perhaps by relaxing our notion of reduction as discussed above, hardness results for these problems may be found. What are the closure properties of the class of polynomially predictable prediction problems?

The negative results based on cryptographic assumptions are disheartening; how low in the complexity hierarchy does one need to go to ensure predictability? Can the negative results of [39], which show that R_{BF} and R_{DFA} are hard based on certain cryptographic assumptions, be proven using the weaker assumption that $\mathcal{RP} \neq \mathcal{NP}$?

Finally, Angluin [3] 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 (or to make other types of queries as described in [5])? If the result $R_{BF} \trianglelefteq R_{DFA}$ of Corollary 5.7 can be proved with an extended notion of reduction allowing membership queries, 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 Eric Allender, Dana Angluin, Sam Buss, David Haussler, Michael Kearns, Nick Littlestone, Yishay Mansour, Ron Rivest, and Allen VanGelder, for helpful discussions concerning this work. We also thank an anonymous referee for a number of excellent suggestions.

References

[1] L. Adleman. Two theorems on random polynomial time. In *Proceedings of the 19th Annual IEEE Symposium on Foundations of Computer Science*, pages 75–83, Washington, D.C., 1978. IEEE Computer Society Press.

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

[3] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987.

[4] D. Angluin. Learning Propositional Horn sentences with hints. Technical Report YALEU/DCS/RR-590, Department of Computer Science, Yale University, December 1987.

[5] D. Angluin. Queries and concept learning. *Machine Learning*, 2:319–342, 1987.

[6] D. Angluin. Equivalence queries and approximate fingerprints. In *Proceedings of the 1989 Workshop on Computational Learning Theory*, pages 134–145, San Mateo, CA, August 1989. Morgan Kaufmann.

[7] D. Angluin and C. H. Smith. Inductive inference: Theory and methods. *Computing Surveys*, 15(3):237–269, 1983.

[8] P. Assouad. Densite et dimension. *Ann. Inst. Fourier*, 33(3):233–282, 1983. Grenoble.

[9] J. M. Barzdin and R. V. Freivald. On the prediction of general recursive functions. *Soviet Mathematics Doklady*, 13:1224–1228, 1972.

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

[11] 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.*

[12] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. Occam's razor. *Information Processing Letters*, 24:377–380, 1987.

[13] R. A. Board and L. Pitt. On the necessity of Occam algorithms. Technical Report UIUCDCS-R-89-1544, Dept. of Computer Science, University of Illinois at Urbana-Champaign, September 1989.

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

[15] A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981.

[16] S. A. Cook. A taxonomy of problems with fast parallel algorithms. *Information and Control*, 64:2–27, 1985.

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

[18] A. Ehrenfeucht and D. Haussler. Learning decision trees from random examples. *Information and Computation*, 82:231–246, 1989.

[19] A. Ehrenfeucht, D. Haussler, M. Kearns, and L. G. Valiant. A general lower bound on the number of examples needed for learning. *Information and Computation*, 82:247–261, 1989.

[20] J. Gill. Probabilistic Turing machines. *SIAM J. Computing*, 6(4):675–695, 1977.

[21] E. M. Gold. Complexity of automaton identification from given data. *Information and Control*, 37:302–320, 1978.

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

[23] O. Goldreich, H. Krawczyk, and M. Luby. On the existence of pseudorandom generators. In *Proceedings of the 29th Annual IEEE Symposium on Foundations of Computer Science*, pages 12–24, Washington, D.C., October 1988. IEEE Computer Society Press.

[24] L. M. Goldschlager. Epsilon-productions in context-free grammars. *Acta Informatica*, (16):303–308, 1981.

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

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

[27] 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*.

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

[29] D. Haussler. Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. *Artificial Intelligence*, (36):177–221, 1988.

[30] D. Haussler, M. Kearns, N. Littlestone, and M. K. Warmuth. Equivalence of models for polynomial learnability. In *Proceedings of the 1988 Workshop on Computational Learning Theory*, pages 42–55, San Mateo, CA, August 1988. Morgan Kaufmann. To appear in *Information and Computation*.

[31] D. Haussler, N. Littlestone, and M. K. Warmuth. Predicting {0,1} functions on randomly drawn points. In *Proceedings of the 29th Annual IEEE Symposium on Foundations of Computer Science*, pages 100–109, Washington, D.C., October 1988. IEEE Computer Society Press.

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

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

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

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

[36] M. Kearns. The computational complexity of machine learning. Technical Report TR-13-89, Ph.D. Thesis, Aiken Computation Laboratory, Harvard University, 1989.

[37] 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 Computing*, New York, May 1987. ACM.

[38] M. Kearns, M. Li, L. Pitt, and L. G. Valiant. Recent results on Boolean concept learning. In *Proceedings of the 4th International Workshop on Machine Learning*, pages 337–352, San Mateo, California, June 1987. Morgan Kaufmann.

[39] M. Kearns and L. G. Valiant. Cryptographic limitations on learning Boolean formulae and finite automata. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, pages 433–444, New York, May 1989. ACM.

[40] R. E. Ladner. The circuit value problem is log space complete for P. *SIGACT News*, 7(1):18–20, 1975.

[41] L. A. Levin. One-way functions and pseudorandom generators. *Combinatorica*, 7(4):357–363, 1987.

[42] M. Li and U. Vazirani. On the learnability of finite automata. In *Proceedings of the 1988 Workshop on Computational Learning Theory*, pages 359–370, San Mateo, California, August 1988. Morgan Kaufmann.

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

[44] L. Pitt. Inductive Inference, DFAs, and Computational Complexity. In *Proceedings of AII-89 Workshop on Analogical and Inductive Inference; Lecture Notes in Artificial Intelligence 397*, pages 18–44, Heidelberg, October 1989. Springer-Verlag.

[45] L. Pitt and L. G. Valiant. Computational limitations on learning from examples. *J. ACM*, 35(4):965–984, 1988.

[46] L. Pitt and M. K. Warmuth. The minimum consistent DFA problem cannot be approximated within any polynomial. Technical Report UIUCDCS-R-89-1499, University of Illinois at Urbana-Champaign, February 1989. To appear, *J. ACM*. A preliminary version appears in the 21st annual ACM Symposium on Theory of Computing, May 1989.

[47] S. J. Russell. Tree-structured bias. In *Proceedings of the AAAI-88*, San Mateo, CA, August 1988. Morgan Kaufmann.

[48] W. L. Ruzzo. On uniform circuit complexity. *J. Comp. Sys. Sci.*, 22:365–383, 1981.

[49] J. E. Savage. *The Complexity of Computing.* John Wiley, New York, 1976.

[50] R. Schapire. The strength of weak learnability. In *Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science*, Washington, D.C., October 1989. IEEE Computer Society Press.

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

[52] I. H. Sudborough. On the tape complexity of deterministic context free languages. *J. ACM*, 25:405–414, 1978.

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

[54] 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.

[55] M. K. Warmuth. Towards representation independence in PAC-learning. In *Proceedings of AII-89 Workshop on Analogical and Inductive Inference; Lecture Notes in Artificial Intelligence 397*, pages 78–103, Heidelberg, October 1989. Springer-Verlag.

[56] Ingo Wegener. *The Complexity of Boolean Functions.* Wiley-Teubner Series in Computer Science, Stuttgart, 1987.

[57] A. C. Yao. Theory and applications of trapdoor functions. In *Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science*, pages 80–91, Washington, D.C., November 1982. IEEE Computer Society Press.

[58] C. K. Yap. *Theory of Complexity Classes.* Oxford University Press, Oxford. To appear.