Machine Learning, 30, 57–87 (1998)
© 1998 Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

Hardness Results for Learning First-Order Representations and Programming by Demonstration

WILLIAM W. COHEN wcohen@research.att.com
AT&T Labs—Research, 180 Park Avenue, Florham Park, NJ 07932

Editor: Dana Ron

Abstract. Learning from "structured examples" is necessary in a number of settings, including inductive logic programming. Here we analyze a simple learning problem in which examples have non-trivial structure: specifically, a learning problem in which concepts are strings over a fixed alphabet, examples are deterministic finite automata (DFAs), and a string represents the set of all DFAs that accept it. We show that solving this "dual" DFA learning problem is hard, under cryptographic assumptions. This result implies the hardness of several other more natural learning problems, including learning the description logic CLASSIC from subconcepts, and learning arity-two "determinate" function-free Prolog clauses from ground clauses. The result also implies the hardness of two formal problems related to the area of "programming by demonstration": learning straightline programs over a fixed operator set from input-output pairs, and learning straightline programs from input-output pairs and "partial traces".

Keywords: computational learning theory, learning deterministic finite automata, inductive logic programming

1. Introduction

In a number of settings it is necessary to learn from "structured" examples: *i.e.*, examples that cannot be easily encoded as feature vectors. Examples of such settings include multiple-instance learning (Dietterich et al., 1997), learning knowledge representation languages (Cohen & Hirsh, 1994b), and inductive logic programming (Quinlan, 1990, De Raedt, 1995, Muggleton & De Raedt, 1994). In this paper we will analyze a simple instance of a learning problem in which examples have a non-trivial structure—specifically, a learning problem in which the examples are directed graphs.

More formally, we consider a learning problem in which the examples are deterministic finite automata (DFAs), the concepts are strings over a fixed alphabet, and a string s denotes the set of all DFAs that accept it. This problem is, in a very natural sense, the dual of the well-investigated problem of learning DFAs from strings[1] (Angluin, 1987, Kearns & Valiant, 1989). Note that in this "dual" DFA learning problem, although the examples have non-trivial structure, the concepts are very simple: a concept essentially traverses a single path through the graph, and tests a single label associated with the endpoint of this path.

We investigate the *polynomial predictability* (Pitt & Warmuth, 1990) of this learning problem: in other words, we investigate the complexity of finding a hypothesis which is probably approximately correct, without placing any restrictions on how the hypothesis is represented. We show that solving the "dual DFA problem" in this representation-independent sense is as hard as solving certain cryptographic problems that are widely assumed to be computationally difficult, such as inverting the RSA encryption function. Moreover, this result holds even if the class of example DFAs is highly restricted.

This result leads immediately to a number of similar hardness results for less artificial non-propositional representations, including the resolution of two previously open problems. As one corollary of the dual DFA result we show that the description logic CLASSIC is not polynomially predictable, again under cryptographic assumptions. *Description logics* are a family of representation languages that have been heavily investigated by the knowledge representation community; for surveys of this work, see Borgida (Borgida, 1992), MacGregor (MacGregor, 1991), or Woods and Schmolze (Woods & Schmolze, 1992). Previously, Cohen and Hirsh (Cohen & Hirsh, 1994a) showed that the description logic CLASSIC is not pac-learnable from examples, and Frazier and Pitt (Frazier & Pitt, 1996) showed that CLASSIC is not pac-learnable from membership queries alone. Both of these negative results, however, pertain only to learners that are restricted to output a hypothesis in the CLASSIC language; the question of the learnability of CLASSIC and related description logics from random examples in a representation independent sense has until now remained open.

Another well-studied problem is the learnability of logic programs (Page & Frisch, 1992, Džeroski et al., 1992, Cohen & Page, 1995). In this paper we show that arity-two "determinate" function-free Prolog clauses are not polynomially predictable, under cryptographic assumptions. Again, although Kietz (Kietz, 1993) showed earlier that this language is not pac-learnable, its learnability in the polynomial predictability model has remained open.

Finally, the dual DFA result gives some insight into the problem of learning simple programs from examples and traces. We show that learning straight-line code (without loops or branches) from input/output pairs is hard, even if there are only three possible actions to take at each step of the program. We then show that this problem becomes tractable if the learner also has access to a "trace" that reveals the intermediate values computed by the target program after performing each individual action. However, learning from traces is shown to be cryptographically hard if an adversary is allowed to hide even $O(\log\log n)$ bits of each intermediate value. Further, learning from traces is shown to be as hard as learning DNF if an adversary is allowed to hide only *two* bits of each intermediate value. These results are motivated by problems from the research area of "programming by demonstration" (Cypher, 1993).

In the remainder of the paper, we will first present some preliminary definitions, and then the hardness results for the dual DFA problem. We will then discuss the technical implications of this result with respect to problems in first-order learning and automatic programming. We will conclude with a summary, and a more general discussion of the consequence of the results.

2. Preliminaries

This paper uses the model of *pac-learnability*, as introduced by Valiant (Valiant, 1984). Let X be a set, called the *domain*. Define a *concept* C over X to be a subset of X, and a *language* $\mathcal{L}$ to be a set of concepts. Associated with $\mathcal{L}$ is some scheme for representing the concepts in $\mathcal{L}$. In general, we will be casual about the distinction between a concept and its representation; when there is a risk of confusion we will write the set denoted by a representation C (*i.e.*, the extension of C) as $ext(C)$. We will assume a *size* or *complexity* *measure* on representations $C \in \mathcal{L}$, and also a size measure on instances $x \in X$. The size of $C \in \mathcal{L}$ (respectively $x \in X$) will be denoted $\|C\|$ (or $\|x\|$). Typically these measures will be polynomially related to the number of bits needed to encode a concept (or instance).

If P is a probability distribution, a *sample of C drawn from X according to P* is a pair of multisets S^+, S^- drawn according to P, S^+ containing only the positive examples of C, and S^- containing the negative ones. We define a sample to be *n_I-bounded* if it contains no example larger than n_I. We will not assume that n_I is known to the learning algorithms.

Informally, pac-learnability requires that a learning algorithm PACLEARN be "probably approximately correct"; *i.e.*, that PACLEARN outputs an accurate hypothesis from a given language $\mathcal{L}$ most of the time, whenever the target concept is succinctly expressible in $\mathcal{L}$. Formally, we define a language $\mathcal{L}$ to be *pac-learnable* iff there is an algorithm PACLEARN and a polynomial function $m(\frac{1}{\epsilon}, \frac{1}{\delta}, n_I, n_T)$ so that for every $n_T > 0$, every $C \in \mathcal{L}$ of size less than n_T, every $0 < \epsilon < 1$, every $0 < \delta < 1$, and every probability distribution P, PACLEARN has the following behavior: when run on a n_I-bounded sample S^+, S^- of C drawn according to P of size $|S^+| + |S^-| > m(\frac{1}{\epsilon}, \frac{1}{\delta}, n_I, n_T)$, PACLEARN outputs a hypothesis $H \in \mathcal{L}$ such that $Prob(P(H \triangle C) > \epsilon) < \delta$, where $\triangle$ denotes symmetric difference, and furthermore, PACLEARN runs in time polynomial in $\frac{1}{\epsilon}$, $\frac{1}{\delta}$, n_I, n_T, and the size of the sample. The probability above is taken over the possible samples S^+ and S^- and (if PACLEARN is a randomized algorithm) over any coin flips made by PACLEARN.

The function $m(\frac{1}{\epsilon}, \frac{1}{\delta}, n_I, n_T)$ is called the *sample complexity* of PACLEARN. A hypothesis H such that $Prob(P(H \triangle C) > \epsilon) < \delta$ is called *ϵ-good with respect to the target C*.

The definition of pac-learnability requires that the hypothesis H of the learner be expressed in the language $\mathcal{L}$. Since this is not always strictly necessary, it is often desirable to relax this requirement (particularly when proving negative results.) We will say that $\mathcal{L}$ is *polynomially predictable* if there is an algorithm PACPREDICT that satisfies all the requirements for a pac-learning algorithm for $\mathcal{L}$, except that PACPREDICT outputs a polynomial time evaluable hypothesis H which is perhaps *not* in the target language $\mathcal{L}$. Negative results in the polynomial predictability model are sometimes called "representation independent" hardness results.

One important analytic tool used in this paper is *prediction-preserving reducibility*, as described by Pitt and Warmuth (Pitt & Warmuth, 1990). If $\mathcal{L}_1$ is a language over domain X_1 and $\mathcal{L}_2$ is a language over domain X_2, then we say that *predicting $\mathcal{L}_1$ reduces to predicting $\mathcal{L}_2$*, written $\mathcal{L}_1 \trianglelefteq \mathcal{L}_2$, iff there is a function $f : X_1 \rightarrow X_2$, henceforth called the *instance mapping*, and a function $g : \mathcal{L}_1 \rightarrow \mathcal{L}_2$, henceforth called the *concept mapping*, so that the following all hold:

1. $x \in C$ if and only if $f(x) \in g(C)$—*i.e.*, concept membership is preserved by the mappings;
2. the size complexity of $g(C)$ is polynomial in the size complexity of C—*i.e.*, the size of concepts is preserved within a polynomial factor; and
3. $f(x)$ can be computed in polynomial time.

Intuitively, $g(C_1)$ returns a concept $C_2 \in \mathcal{L}_2$ that will "emulate" C_1 (*i.e.*, make the same decisions about concept membership) on examples that have been "preprocessed" with the function f. Pitt and Warmuth (Pitt & Warmuth, 1990) showed that if $\mathcal{L}_1 \trianglelefteq \mathcal{L}_2$ and $\mathcal{L}_2$ is polynomially predictable, then $\mathcal{L}_1$ is also polynomially predictable. Conversely, if $\mathcal{L}_1 \trianglelefteq \mathcal{L}_2$ and $\mathcal{L}_1$ is not polynomially predictable, then neither is $\mathcal{L}_2$.

3. Hardness of the dual DFA problem

A well-studied problem in computational learning theory is the learnability of DFAs from strings (Angluin, 1987, Kearns & Valiant, 1989). We will now consider a dual version of this problem, in which the concepts are strings and the examples are DFAs. Let us consider the domain of DFAs over a fixed alphabet Σ, and the concept class $\mathcal{S}^{\mathrm{DFA}}$ of strings $s \in \Sigma^*$, with semantics defined as follows: if M is a DFA and s is a string in Σ^*, then $M \in ext(s)$ iff s is accepted by M. In other words, a string denotes the class of DFAs that accept it.

Let us first consider the learnability of this language in the pac-learning model. Using a construction from Cohen and Hirsh (Cohen & Hirsh, 1994a) it can be easily shown that the dual DFA learning problem is hard in the pac-learnability model.

THEOREM 1 *The language $\mathcal{S}^{\mathrm{DFA}}$ is not pac-learnable unless RP=NP.*

Proof: We only sketch the argument, as it closely parallels (part of) the argument used in Theorem 3 of Cohen and Hirsh (Cohen & Hirsh, 1994a). By the results of Pitt and Valiant (Pitt & Valiant, 1988), a language is pac-learnable only if there is a polynomial-time algorithm for the corresponding *consistency problem*—in this case, the problem of finding a consistent hypothesis in $\mathcal{S}^{\mathrm{DFA}}$ given a set of positive and negative example DFAs. We will reduce 3SAT (Hopcroft & Ullman, 1979) to the consistency problem for dual DFAs, thereby showing that the consistency problem is NP-hard.

Assume that there is an algorithm A that solves the consistency problem, and let $\phi = \bigwedge_{i=1}^{n}(l_{i_1} \vee l_{i_2} \vee l_{i_3})$ be a 3CNF sentence over n variables, where each literal l_{i_j} is either a variable x_{i_j} or its negation. Without loss of generality, assume that the literals l_{i_1}, l_{i_2}, and l_{i_3} are in strictly increasing alphabetical order. Now construct from ϕ a set of n DFAs $M_1, \ldots, M_n$ where M_i is the minimal DFA accepting the language

$$\begin{array}{l}(0+1)^{i_1-1}s(l_{i_1})(0+1)^{n-i_1} \;\; + \\ (0+1)^{i_2-1}s(l_{i_2})(0+1)^{n-i_2} \;\; + \\ (0+1)^{i_3-1}s(l_{i_3})(0+1)^{n-i_3}\end{array}$$

Where $s(l)$ is 1 if $l = x_k$ and $s(l)$ is 0 if $l = \overline{x_k}$. The DFAs $M_1, \ldots, M_n$, are then presented to A as positive examples. Now, if the binary strings of length n accepted by these DFAs are interpreted as assignments to the variables that appear in ϕ, it is easily verified that M_i accepts exactly the assignments that satisfy the i-th clause of ϕ. Thus any concept (string) s that covers all the positive examples must satisfy all the literals of ϕ, and therefore satisfy ϕ itself, and hence if a polynomial time algorithm A exists it can be used to solve instances of 3SAT. ■

Proposition 1 shows that learning is hard when the learner is required to output a hypothesis in the target language $\mathcal{S}^{\mathrm{DFA}}$. Now let us consider the more interesting question of whether a pac hypothesis can be found in some other representation—*i.e.*, the question of whether this language is polynomially predictable.

Table 1. Examples of formulae in $\mathcal{B}^d_{n,*}$ for various values of d

d	$\mathcal{B}^d_{n,*}$
0	$x_1 \wedge \overline{x_3}$
0	$x_2 \wedge \overline{x_4}$
1	$(x_2 \wedge \overline{x_4}) \vee (x_1 \wedge \overline{x_3})$
1	$(x_1 \wedge x_2 \wedge x_3) \vee (\overline{x_1} \wedge \overline{x_2} \wedge \overline{x_3})$
2	$((x_2 \wedge \overline{x_4}) \vee (x_1 \wedge \overline{x_3})) \wedge ((x_4 \wedge \overline{x_2}) \vee (x_3 \wedge \overline{x_1}))$

The principle technical result of this paper is the following theorem, which shows that the "dual DFA problem" is as hard as learning log-depth boolean circuits, even if example DFAs are restricted to be over a three-letter alphabet. In fact, in the proof, we will show that this result holds even for a rather restricted class of DFAs, namely those that are also acyclic, leveled, and of logarithmic level width (as defined below).

THEOREM 2 *If $|\Sigma| \geq 3$ then the language $\mathcal{S}^{\rm DFA}$ is not polynomially predictable under cryptographic assumptions.*[2]

The remainder of this section is a proof of this result. The proof is based on a prediction-preserving reducibility from a certain class of boolean formulae, which we define below, to the dual DFA learning problem. Given this reduction, existing hardness results for boolean formulae can be used to establish the theorem itself.

Given the boolean variables $x_1, \ldots, x_n$, define the class of boolean formulae $\mathcal{B}^d_{n,*}$ inductively as follows. (The reason for the somewhat cumbersome notation will become clear shortly.)

- $\mathcal{B}^0_{n,*}$ is the class of monomials over $x_1, \ldots, x_n$.
- if $d > 0$ and d is odd, then

$$\mathcal{B}^d_{n,*} \equiv \{b_1 \vee b_2 : b_1 \in \mathcal{B}^{d-1}_{n,*} \text{ and } b_2 \in \mathcal{B}^{d-1}_{n,*}\}$$

- if $d > 0$ and d is even, then

$$\mathcal{B}^d_{n,*} \equiv \{b_1 \wedge b_2 : b_1 \in \mathcal{B}^{d-1}_{n,*} \text{ and } b_2 \in \mathcal{B}^{d-1}_{n,*}\}$$

In other words, $\mathcal{B}^d_{n,*}$ is the class of balanced alternating depth-d boolean formulae over n variables with monomials as leaves. Table 1 contains some examples of formulae in $\mathcal{B}^d_{n,*}$.

We will now define some restrictions on DFAs. Let M be a DFA with start state q_0 and transition function δ. Define LEVEL(d, M) to be the set of states in M that can be reached with input strings of length d. The *level width* of a DFA M is defined to be the maximum cardinality over all d of the set LEVEL(d, M). A DFA is *leveled* if LEVEL(d_1, M) and LEVEL(d_2, M) are disjoint for all $d_1 \neq d_2$. Note that leveled DFAs are always acyclic.

We define LDFA(w) to be the set of leveled DFAs of level width at most w. By way of example, Figure 1 shows two leveled DFAs of width 2, and Figure 4 shows a leveled DFA of width 3.

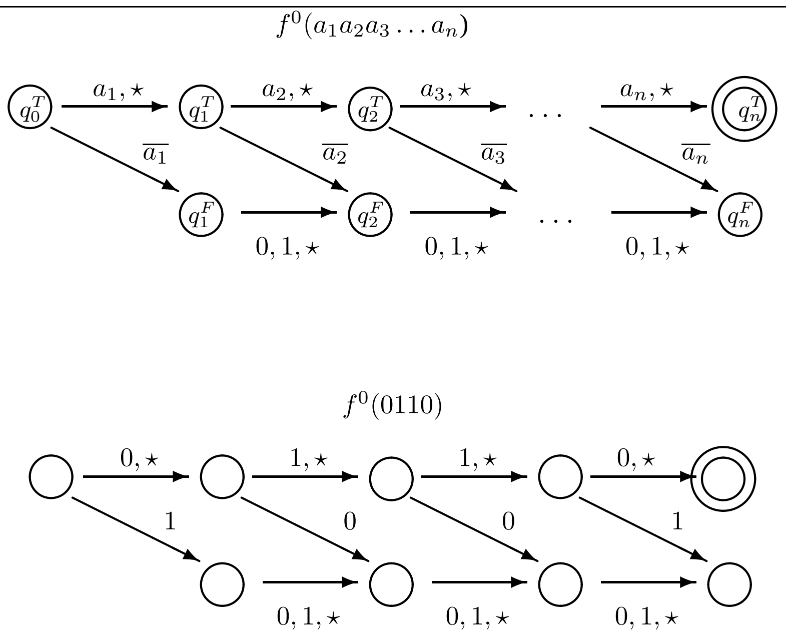


Figure 1. The general construction used for $f^0(\eta)$, and a specific example.

We will also adopt the following notation: if $\mathcal{L}$ is a language over the domain X, then $\mathcal{L}_{n_I,n_T}$ denotes the set of concepts $\{C \in \mathcal{L} : \|C\| \le n_T\}$ over the restricted domain $\{x \in X : \|x\| \le n_I\}$; that is, $\mathcal{L}_{n_I,n_T}$ is the set of small (size $\le n_T$) concepts over the domain of small (size $\le n_I$) instances. Also let $\mathcal{L}_{n_I,*}$ denote the set $\bigcup_j \mathcal{L}_{n_I,j}$. Thus $\mathcal{S}^{\mathrm{LDFA}(w)}_{n_I,n_T}$ denotes the class of all strings of length at most n_T over the domain of width-w leveled acyclic DFAs of size at most n_I. The size measure we will use for instances (DFAs) is simply the number of states.

We will now show that $\mathcal{B}^d_{n,*} \trianglelefteq \mathcal{S}^{\mathrm{LDFA}(d+2)}_{p_I(n,d),p_T(n,d)}$ where the functions $p_I(*,*)$ and $p_T(*,*)$ are both polynomials in n and 2^d. The lemma below establishes a slightly stronger result by induction on d.

Lemma 1 *Assume* $|\Sigma| \ge 3$. *Then for all* $d > 0$ *and all* $n > 2$,

$$\mathcal{B}^d_{n,*} \trianglelefteq \mathcal{S}^{\mathrm{LDFA}(d+2)}_{n_I,n_T}$$

where $n_I = (d+2)(n+2)2^d$ *and* $n_T = (n+2)2^d$. *Further, in every example DFA there is exactly one accepting state, which appears at the maximal depth, and exactly two states total at the maximal depth.*

Proof: The proof is by induction on d. Without loss of generality let $\Sigma = \{0,1,\star\}$. (The symbol "$\star$" will be used as a sort of a wildcard in our construction, and should not be confused with the regular expression "star" operator.) For each d we will produce an instance mapping f^d and a concept mapping g^d that preserve membership, as required by the definition of prediction-preserving reducibilities, and also satisfy the other conditions of the lemma. We will focus on the bounds on level width (of $d+2$) and string length (of $(n+2)2^d$), since the bound on instance size is implied by these bounds.

Base case. Let $\eta = a_1 \ldots a_n$ be an assignment to $x_1, \ldots, x_n$. (*I.e.*, $a_i = 1$ if x_i is true and $a_i = 0$ if x_i is false.) Let $\overline{a_i}$ denote the negation of a_i. (*I.e.*, $\overline{0} = 1$ and $\overline{1} = 0$.) Recall that the instance mapping f here must map an assignment η to a DFA, and define $f^0(\eta)$ to be a DFA with the following structure.

1. Recalling that n is the number of variables involved in the assignment η, there are $2n+1$ states: the start state q_0^T, n states named $q_1^T, \ldots, q_n^T$, and n states named $q_1^F, \ldots, q_n^F$. The only accepting state is q_n^T.

2. For $i : 1 \leq i \leq n$,
   - there is an arc labeled $\overline{a_i}$ from q_{i-1}^T to q_i^F,
   - there are two arcs labeled a_i and $\star$ from q_{i-1}^T to q_i^T, and
   - if $i > 1$, then there are three arcs labeled a_i, $\overline{a_i}$, and $\star$ from q_{i-1}^F to q_i^F.

See Figure 1 for examples of this construction.

Now, we will define the corresponding concept mapping g^0. Recalling that g^0 must map a monomial to a string, let $b \in \mathcal{B}_{n,*}^0$ be a monomial over $x_1, \ldots, x_n$ and define σ_i as follows:

$$\sigma_i \equiv \begin{cases} 1 & \text{if } x_i \in b \\ 0 & \text{if } \overline{x_i} \in b \\ \star & \text{else} \end{cases}$$

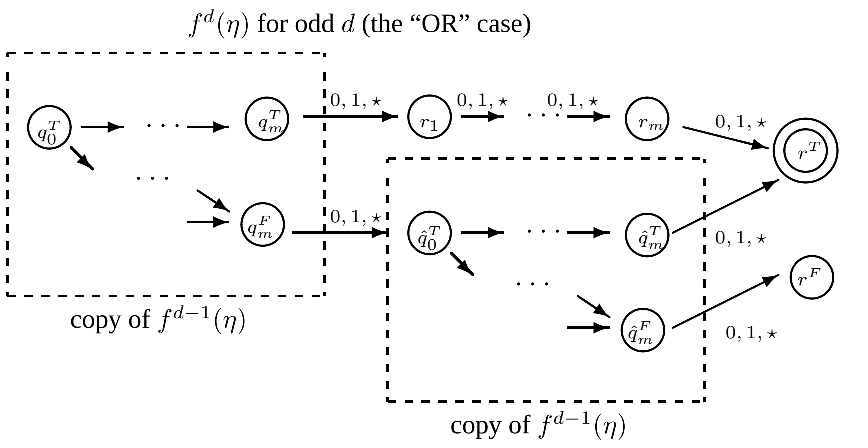
We define $g^0(b)$ to be the string $\sigma_1 \ldots \sigma_n$. For example, when $n = 4$ then

$$g^0(x_2\overline{x_4}) = \star 1 \star 0$$

Let us consider now the size and level width bounds. Clearly, $f^0(\eta)$ is leveled, and the width of $f^0(\eta)$ is exactly $2 = 0 + 2$; also $g^0(b)$ is of size $n < (n+2)2^0$. It is also obvious that f^0 is computable in polynomial time, and that there are two states of maximal depth, exactly one of which is accepting, and no other accepting states in the automaton.

It remains to be shown that membership is preserved. It is not too hard to see that the DFA $f^0(\eta)$ accepts the string $g^0(b)$ exactly when η satisfies b. To argue this formally, we will introduce the following notation. For a string s and a DFA M, let $M(s)$ denote the state reached by M after reading in the string s, and let $M_\eta^0 = f^0(\eta)$. Consider the monomial b as a subset of the literals $\{x_1, \overline{x_1}, \ldots, x_n, \overline{x_n}\}$, and let $b|_j$ denote $b \cap \{x_1, \overline{x_1}, \ldots, x_j, \overline{x_j}\}$. By induction one can easily show that for all j, $M_\eta^0(\sigma_1 \ldots \sigma_j)$ must be either q_j^T or q_j^F, and that $M_\eta^0(\sigma_1 \ldots \sigma_j) = q_j^T$ iff η satisfies $b|_j$. Thus the lemma holds for the base case of $d = 0$.

To summarize the argument above, we have so far shown that learning problem for monomials can be reduced to the dual DFA learning problem. To accomplish this reduction, it was necessary to show that an assignment η can be converted into a DFA which accepts exactly those monomials that are satisfied by η (for a suitable encoding of monomials). As



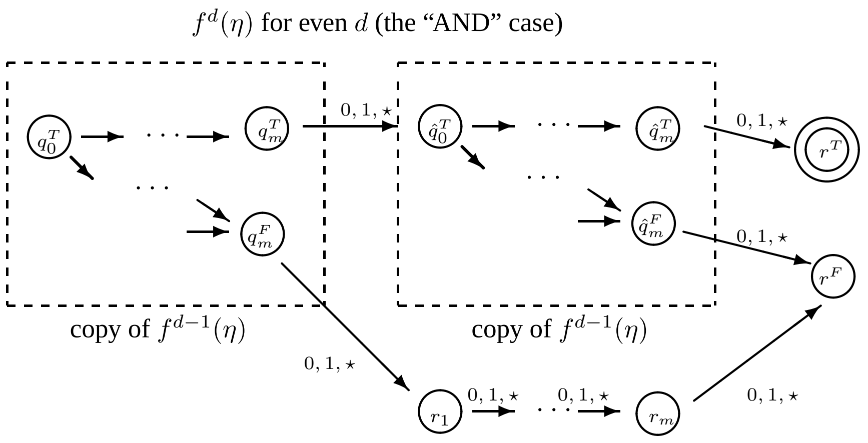


Figure 2. The constructions used for $f^d(\eta)$ for $d > 0$

Figure 1 shows, this computation is easy to do with a DFA. With the encoding we chose, the DFA can simply scan through the literals in the monomial, one by one; it is only necessary for the automaton to "remember" if any literal in the monomial has been falsified by η.

The inductive step. Let us consider first the case in which d is odd—and hence for a formula $b \in \mathcal{B}^d_{n,*}$, $b = b_1 \vee b_2$. By induction there exists an instance mapping f^{d-1} and a concept mapping g^{d-1} that satisfy the conditions of the lemma; also b_1 and b_2 are both in $\mathcal{B}^{d-1}_{n,*}$.

We now define f^d and g^d as follows. For an assignment η, $f^d(\eta)$ is an automaton with the following structure.

1. Let m be the depth of $f^{d-1}(\eta)$. We will use q_0^T to denote the start state of $f^{d-1}(\eta)$, q_m^T to denote the maximal-depth accepting state, and q_m^F to denote the maximal-depth rejecting state. (Note that by induction, q_m^T and q_m^F are unique.)

   The states of the automaton include each state q in $f^{d-1}(\eta)$; a copy $\hat{q}$ of each state q of $f^{d-1}(\eta)$; m states $r_1, \ldots, r_m$; and two additional states r^T and r^F.

   The start state is q_0^T. The sole accepting state is r^T.

2. If there is an arc labeled a in $f^{d-1}(\eta)$ between states q_i and q_j, then there is an arc labeled a in $f^d(\eta)$ between states q_i and q_j, and also between states $\hat{q}_i$ and $\hat{q}_j$.

3. For $i : 1 \leq i < m$ there are arcs labeled 0, 1 and $\star$ from state r_i to state r_{i+1}.

4. There are arcs labeled 0, 1 and $\star$ between all of following pairs of states: q_m^T and r_1, q_m^F and $\hat{q}_0^T$, r_m and r^T, $\hat{q}_m^T$ and r^T, and $\hat{q}_m^F$ and r^F.

When d is even, then conditions 1 through 3 are the same, but condition 4 is amended to require arcs labeled 0, 1 and $\star$ between these pairs of states: q_m^T and $\hat{q}_0^T$, q_m^F and r_1, r_m and r^F, $\hat{q}_m^T$ and r^T, and $\hat{q}_m^F$ and r^F.

See Figure 2 for examples of the construction. Clearly this DFA can be constructed in polynomial time, if the size bounds of the theorem hold.

To define the concept mapping g^d, let $\alpha = g^{d-1}(b_1)$ and $\beta = g^{d-1}(b_2)$. We define $g^d(b_1 \vee b_2)$ (for odd d) or $g^d(b_1 \wedge b_2)$ (for even d) to be the string $\alpha\star\beta\star$. For example, when $n = 4$, then the following are examples of the mapping g^d. (The underlining is for clarity, and shows the recursive structure of the strings.)

$$\begin{aligned} g^0(x_2\overline{x_4}) &= \star 1 \star 0 \star \\ g^1(x_2\overline{x_4} \vee \overline{x_2}x_4) &= \underline{\star 1 \star 0} \star \underline{\star 0 \star 1}\star \\ g^1(x_3\overline{x_4} \vee \overline{x_3}x_4) &= \underline{\star\star 10} \star \underline{\star\star 01}\star \\ g^2((x_2\overline{x_4} \vee \overline{x_2}x_4) \wedge (x_3\overline{x_4} \vee \overline{x_3}x_4)) &= \underline{\underline{\star 1 \star 0} \star \underline{\star 0 \star 1}\star} \star \underline{\underline{\star\star 10} \star \underline{\star\star 01}\star} \star \end{aligned}$$

Below we will argue that this construction correctly implements boolean AND and OR for functions in $\mathcal{B}_{n,*}^d$; that is, we will argue that the DFA $f^d(\eta)$ for odd d accepts exactly the formulae of the form $b_1 \vee b_2$ satisfied by η, and that $f^d(\eta)$ for even d accepts exactly the formulae of the form $b_1 \wedge b_2$ satisfied by η. By induction we can assume that the DFA $f^{d-1}(\eta)$ accepts exactly those formulae satisfied by η. Again, the basic idea is simple. The constructed DFA scans the encoding of b_1, and then the encoding of b_2. In each case it is only necessary for the automaton to "remember" which of the subformulae b_1 and b_2 are satisfied by η to correctly perform the computation; and by induction, copies of the automaton for $f^{d-1}(\eta)$ can be used to determine if η satisfies the subexpressions b_1 and b_2.

More formally, we wish to show that for any assignment η and any $b \in \mathcal{B}_{n,*}^d$, η satisfies b iff $f^d(\eta)$ accepts $g^d(b)$. Assume that this is true for $d-1$, let M_η^d be the machine $f^d(\eta)$, and let $M(s)$ denote the state that the DFA M is in after reading in the string s. We can now argue as follows.

Case 1. Assume d is odd. If η satisfies $b = b_1 \vee b_2$, either η satisfies b_1 or b_2 or both.
For the following, please refer to Figure 2. If η satisfies b_1 then by the inductive

hypothesis $M_\eta^{d-1}(g^{d-1}(b_1))$ is the accepting state q_m^T; referring to Figure 2, clearly $M_\eta^d(g^d(b)) = r^T$, and hence $g^d(b)$ is accepted by M_η^d. If η satisfies b_2 but not b_1 then $M_\eta^{d-1}(g^{d-1}(b_1))$ is the rejecting state q_m^F, but $M_\eta^{d-1}(g^{d-1}(b_2))$ is the accepting state q_m^T. Again, $M_\eta^d(g^d(b)) = r^T$, following the path $q_0^T, \ldots, q_m^F, \hat{q}_0^T, \ldots, \hat{q}_m^T, r^T$.

Conversely, suppose η does not satisfy $b = b_1 \vee b_2$. Then η satisfies neither b_1 nor b_2. By induction M_η^{d-1} rejects both $g^{d-1}(b_1)$ and $g^{d-1}(b_2)$, and hence the automaton M_η^d will, on reading the string $g^d(b)$, visit the states $q_0^T, \ldots, q_m^F, \hat{q}_0^T, \ldots, \hat{q}_m^F$, and finally r^F, rejecting the string.

Case 2. Assume d is even. The argument is analogous to Case 1. If η satisfies $b = b_1 \wedge b_2$, then M_η^{d-1} accepts $g^{d-1}(b_1)$ and $g^{d-1}(b_2)$. Hence M_η^d will, on reading the string $g^d(b)$, visit the states $q_0^T, \ldots, q_m^T, \hat{q}_0^T, \ldots, \hat{q}_m^T$, and finally r^T, accepting the string. If η does not satisfy b, then M_η^{d-1} rejects either $g^{d-1}(b_1)$ or $g^{d-1}(b_2)$. In either case M_η^d finally reaches the state r^F, rejecting the string $g^d(b)$.

Finally let us consider the size and level width bounds. Inspection of the construction shows that the level width of $f^d(\eta)$ is bounded by one plus the level width of $f^{d-1}(\eta)$. By induction, this can be bounded by $((d-1)+2)+1 = d+2$. For the size bound on $g^d(b)$, let LEN(d) denote the length of $g^d(b)$ for $b \in \mathcal{B}_{n,*}^d$. We claim that for all d

$$\mathrm{LEN}(d) \equiv n2^d + \sum_{i=1}^{d} 2^i$$

If true, this claim clearly satisfies the size bound stated in the lemma, since $\sum_{i=1}^{d} 2^i < (n+2)2^d$. The base case for the claim is immediate, as LEN$(0) = n$. The inductive case can be easily verified by substitution, using the fact that LEN$(d) = 2$LEN$(d-1)+2$:

$$\begin{aligned}
\mathrm{LEN}(d) &= 2 \cdot (\mathrm{LEN}(d-1)+1) \\
&= 2 \cdot (n2^{d-1} + \sum_{i=1}^{d-1} 2^i + 1) \\
&= n2^d + \sum_{i=1}^{d-1} 2^{i+1} + 2 \\
&= n2^d + \sum_{i=1}^{d} 2^i
\end{aligned}$$

This completes the proof of the lemma. ■

Lemma 1 gives a polynomial reduction from $\mathcal{B}_{n,*}^{\log n}$ to the dual DFA problem. To complete the proof of Theorem 2, it is only necessary to show that $\mathcal{B}_{n,*}^{\log n}$ is cryptographically hard. This result follows easily from known results on circuit complexity (Boppana & Sipser, 1990); however for completeness, we will sketch the argument.

Consider the language of boolean circuits using AND, OR and unary NOT gates with fan-in two and unbounded fan-out. Any boolean circuit can be converted to a boolean formula by replicating portions of the circuit; note that if this is done, a gate at level d need be replicated at most 2^d times. This means that circuits of depth $\log n$ can be converted to boolean formulae of depth $\log n$ with only a polynomial increase in size—specifically the size is increased by a factor of n. Also note that negations appearing in a boolean formula can be pushed to the inputs by repeated application of De Morgan's laws, without any increase in size. Finally, a boolean formula that contains only AND and OR operators internally can be forced to strictly alternate AND and OR operators by padding non-alternating subformulae. (For example, AND(AND(w,x),OR(y,z)) would be replaced by AND(OR(AND(w,x),AND(w,x)),OR(y,z)).) Note that padding will at most double the depth of the formula. Thus log-depth circuits can be represented as log-depth strictly alternating boolean formulae—a strict subset of $\mathcal{B}_{n,*}^{\log n}$.

Thus we have the following proposition.

PROPOSITION 1 *For every boolean circuit C of depth $\log n$ over n variables, there is an equivalent formula C' in $\mathcal{B}_{n,*}^{2\log n}$.*

By Lemma 1 the size of C' is bounded by $(n+2)2^{2\log n} = (n+2)n^2$. Thus together with the reduction of Lemma 1 this proposition shows that the language of log-depth circuits is prediction-preserving reducible to

$$\mathcal{S}^{\mathrm{LDFA}(2\log n+2)}_{(2\log n+2)(n+2)n^2,(n+2)n^2}$$

which is polynomial in n. The expressive power of depth-bounded boolean circuits, as well as their learnability, has been well studied; in particular it is known that log-depth circuits are hard to predict under cryptographic assumptions (Kearns & Valiant, 1989, Theorem 4).[3]

This completes the proof of Theorem 2. Note that the construction actually shows the dual DFA problem to be cryptographically hard even for a rather restricted class of DFAs: the examples used in the construction are all acyclic, leveled, and of logarithmic level width.

In passing, we note that for the hardness result above, it would be sufficient to consider a further restriction of $\mathcal{B}_{n,*}^{d}$, in which the leaves are single variables rather than monomials. The constructions and the proof for this simpler class of boolean formulae would be essentially identical to the proof above; we have chosen the slightly more complex class $\mathcal{B}_{n,*}^{d}$ for this reduction because it will simplify the proof of Theorem 8, below.

4. The dual DFA result and first-order learnability

The dual DFA problem is an interesting but somewhat artificial problem. In the introduction, we motivated analysis of the dual DFA problem based on its broad similarity to problems such as relational learning and inductive logic programming; like these problems, the dual DFA examples have non-trivial structure (namely, a graph-like structure.)

In this section we will discuss some more concrete relationships between the dual DFA problem and certain learnability problems for first-order languages. In particular, we will show that the dual DFA learning problem can be easily reduced to two previously open learning problems, one involving a restricted class of logic programs, and one involving description logics. The reductions show these learning problems to be hard, under cryptographic assumptions.

```
(AND WOMAN
     (ALL DAUGHTER
          (AND THEORYPHD
               UNEMPLOYED))
     (ALL SON
          (AND MARRIED
               (ALL SPOUSE
                    DOCTOR))))
```

"The set of women whose daughters are all unemployed theory PhDs, and all of whose sons are married to doctors."

```
(AND MORTGAGEAPPLICATION
     (SAME-AS (GUARANTOR)
              (APPLICANT SPOUSE
                         MOTHER)))
```

"The set of mortgage applications that are guaranteed by the applicant's mother-in-law."

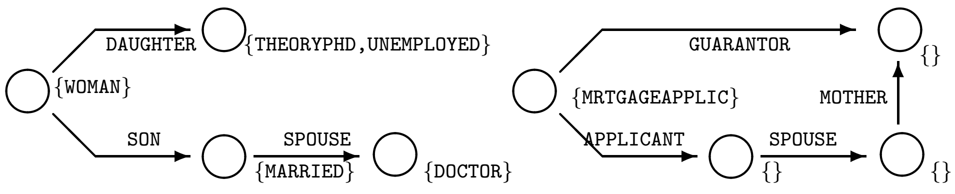


Figure 3. Example CoreClassic concepts, and equivalent concept graphs

It should be emphasized that although the dual DFA problems is closely connected with first-order learning, the dual nature of the problem makes our results incomparable to results obtained in previously studied graph-learning problems (Angluin, 1988, Ergün et al., 1995)—in these problems, the *hypothesis* space, rather than the instance space, is a set of graphs.

4.1. Description logics with equality

4.1.1. Background. *Description logics* or *terminological logics* are a family of knowledge representation and reasoning systems that have found applications in several diverse areas, ranging from database interfaces (Beck et al., 1989) to software information bases (Devanbu et al., 1991) to financial management (Mays et al., 1987) to hardware configuration (Wright et al., 1993). Most of the applications of description logics have not involved learning; however, the learnability of description logics has also been analyzed (Cohen & Hirsh, 1994a, Cohen & Hirsh, 1994b, Frazier & Pitt, 1996). In this section we will consider the pac-predictability of CoreClassic, the simple description logic analyzed by Cohen and Hirsh (Cohen & Hirsh, 1994a).

Briefly, description logics are to used to reason about *descriptions*, which describe sets of atomic elements called *individuals*. Individuals can be organized into *primitive classes*, which denote sets of individuals, and are related through binary relations called *roles* (or *attributes* when the relation is functional). For example, the individuals DR-JOHNSON and

`CS-101` might be related by the `TEACHES` role, and `CS-101` might be an instance of the primitive class `COURSE`. *Descriptions* are composite terms that denote sets of individuals, and are built from primitive classes (such as `PERSON`), and restrictions on the properties an individual may have, such as the kinds or number of role fillers. For instance the description

```
(AND PERSON (ALL TEACHES (AND GRADUATE-LEVEL COURSE)))
```

might denote "the set of people that teach only graduate-level courses", or in predicate calculus, the set of individuals x that satisfy

$$\texttt{PERSON}(x) \wedge \forall y[\texttt{TEACHES}(x, y) \Rightarrow (\texttt{GRADUATE-LEVEL}(y) \wedge \texttt{COURSE}(y))]$$

CORECLASSIC is a description logic containing primitive concepts, roles, attributes, and the constructors `AND`, `ALL`, and `SAME-AS`. The `SAME-AS` constructor is used to require that the result of following two chains of attributes will lead to the same individual: for instance the description

```
(AND COURSE (SAME-AS (INSTRUCTOR) (PRINCIPLE-TEXT AUTHOR)))
```

might denote "the set of courses where the instructor is the author of the principle textbook", or in predicate calculus, the set of individuals x such that

$$\texttt{COURSE}(x) \wedge [\texttt{INSTRUCTOR}(x) = \texttt{AUTHOR}(\texttt{PRINCIPLE-TEXT}(x))]$$

Some additional examples of CORECLASSIC descriptions are shown in Figure 3, and for readers unfamiliar with description logics, Appendix A gives a brief overview of the semantics for the language. More detailed descriptions can be found elsewhere (Borgida & Patel-Schneider, 1994).

An important operation in description logics is determining if a *subsumption* relationship holds between two concepts. Roughly speaking, concept C_1 subsumes concept C_2 if C_1 is more general than C_2. (See Appendix A for more discussion.) Cohen and Hirsh (Cohen & Hirsh, 1994a) and Frazier and Pitt (Frazier & Pitt, 1996) have investigated the learnability of CORECLASSIC and similar description logics in a setting in which examples are concepts, marked as positive if and only if they are subsumed by the target concept.[4]

Cohen and Hirsh showed that CORECLASSIC is not pac-learnable in this model, but that a version of CORECLASSIC allowing only restricted use of SAME-AS is pac-learnable. A later paper (Cohen & Hirsh, 1994b) presented additional formal results for a more expressive description logic, and experimental results for a number of learning problems. Frazier and Pitt (Frazier & Pitt, 1996) showed that CORECLASSIC is not exactly learnable from membership queries alone, but is exactly learnable from membership and equivalence queries. They also demonstrated that the more expressive logic CLASSIC is learnable from membership and equivalence queries.

4.1.2. A representation-independent negative result. Although previous results identify several cases in which pac-learning CORECLASSIC is difficult, the question of the pac-predictability of CORECLASSIC in representation independent models has, until now, remained open.[5] However, the following result is a corollary of Theorem 2.

THEOREM 3 *Under the cryptographic assumptions of Theorem 2,* CORECLASSIC *is not polynomially predictable in the model in which examples are concepts labeled by their subsumption relationship with the target concept.*

Proof: An important tool in the analysis of CORECLASSIC is the notion of a *concept graph* (Borgida & Patel-Schneider, 1994). A concept graph is a directed rooted graph in which the arcs are labeled by roles and attributes and the nodes are labeled by primitives. SAME-AS restrictions are represented as multiple paths to a node. Figure 3 contains two examples of concept graphs; in each case, the leftmost node in the Figure is the root of the graph. Notice that, at least superficially, concept graphs resemble finite automata: they are both rooted directed graphs with labeled nodes and edges. A concept graph is said to be *well-formed* if certain other conditions are met, including a restriction that states that for every node of the graph and every possible arc label a, there will be at most one outgoing edge labeled a. Notice that well-formedness makes concept graphs resemble (at least superficially) *deterministic* finite automata.

It should perhaps be noted that without the SAME-AS construct, CORECLASSIC concept graphs are always trees, rather than arbitrary graphs. Most of the complexities in description logic learnability arise from the graph-like nature of descriptions; learning algorithms for tractable description logics are often fairly simple to implement if the SAME-AS construct is disallowed.

A detailed description of the semantics of CORECLASSIC concept graphs is beyond the scope of this paper. For our purposes, the following two facts are critical.

- For every DFA M over the alphabet Σ there is a well-formed concept graph G over the primitive alphabet $\{acc\}$, role alphabet $\emptyset$ and attribute alphabet Σ such that $a_1 \ldots a_n \in L(M)$ if and only if the concept (ALL a_1 (ALL $a_2 \ldots$ (ALL a_n *acc*) $\ldots$)) subsumes G. Furthermore, G can be constructed from M in polynomial time (Cohen & Hirsh, 1994a, Proposition 1).

- The language of CORECLASSIC and concept graphs are equivalent up to polynomial factors: that is, for every CORECLASSIC concept D, there is a semantically equivalent well-formed concept graph of size polynomial in $\|D\|$ that can be constructed in polynomial time, and for every well-formed concept graph G, there is a semantically equivalent CORECLASSIC description of size polynomial in $\|G\|$ that can be constructed in polynomial time (Cohen & Hirsh, 1994a, Theorem 1).

From this it is clear that prediction-preserving reducibilities exist from S^{DFA} to CORECLASSIC concept graphs to CORECLASSIC. Specifically, one can map example DFAs to concept graphs and thence to CORECLASSIC concepts, and one can map a string $s = a_1 \ldots a_n$ to concepts of the form (ALL a_1 (ALL $a_2 \ldots$ (ALL a_n *acc*). ■

Recall that we have shown the hardness of the dual DFA problem for a restricted case: three-letter alphabets and acyclic, leveled automata. Similar restrictions can be applied to CORECLASSIC to show that it is hard to learn learn (in the polynomial predictability model) if there are only three attributes, and if concept graphs must be acyclic.

4.1.3. Another recent negative result. Another recent negative result on learning CoreClassic is due to Frazier and Pitt (Frazier & Pitt, 1996). They consider a different learning model in which concepts are learned from individuals. To motivate this model, we note that description logics are often used in conjunction with a large knowledge base of "assertions", or facts about individuals. This knowledge base can be visualized as a large graph in which the nodes correspond to individuals (which are atomic entities) and the arcs correspond to roles (binary relationships) that relate the individuals. The knowledge base thus corresponds roughly to a concept graph, except that it is typically far larger, and has no distinguished root node.

Frazier and Pitt considered a model where individuals in the knowledge base are examples. Roughly speaking, an individual I will be labeled positive for a target concept C if C subsumes the concept graph obtained by making the node of the knowledge base graph corresponding to I the root. In Frazier and Pitt's model, the knowledge base is also assumed to be extremely large—potentially exponential in the size of the target concept. Under these assumptions, learning a CoreClassic concept is as hard as learning polynomial sized circuits, and hence is cryptographically hard.

The Frazier and Pitt model of learning from individuals and a large knowledge base is rather different from the model of learning from concepts, and hence their result is not directly comparable to ours. One important difference is that in learning from subsumed concepts, the training examples all are of polynomial size; however, in learning from individuals, there is an exponential amount of information about each individual that is potentially relevant to a learning problem. In particular, the restrictions associated with any path from that individual to any other node in the (exponentially large) knowledge base could potentially be included in the target concept.

Frazier and Pitt's hardness result also requires a polynomial number of attributes, whereas our result requires only a constant number of attributes. However the Frazier and Pitt result requires weaker cryptographic assumptions.

4.1.4. A positive result for the dual DFA problem. Frazier and Pitt (Frazier & Pitt, 1996) also showed that Classic (a generalization of CoreClassic) is exactly learnable from membership and equivalence queries. This result, together with the observations above, suggests that $\mathcal{S}^{\mathrm{DFA}}$ might also be learnable from membership and equivalence queries. This is the case; in fact, $\mathcal{S}^{\mathrm{DFA}}$ can be shown to be learnable from membership queries alone.

Define a *membership query* to be a query to an oracle in which the oracle is supplied with an instance M, and answers "positive" if M is a member of the target concept s, and "negative" otherwise. We have the following result.

Theorem 4 *$\mathcal{S}^{\mathrm{DFA}}$ is learnable from membership queries.*

Proof: Let $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$ be the alphabet, and consider the following algorithm, which uses a series of queries to find the letters of the target string in left-to-right order.

1. Let h be the empty string.
2. Issue k queries using the minimal DFAs accepting the languages $h\sigma_1\Sigma^*, \ldots, h\sigma_k\Sigma^*$.

3. If there is exactly one "positive" answer from these queries, let $h = h\sigma_i$, where σ_i was the letter associated with the sole positive example, and go to Step 2.

4. If none of the queries is answered "positive", then output h and halt.

To establish the correctness of this procedure, observe first that at Step 2, if h is a proper prefix of the target string s, then exactly one of the queried DFAs can accept s. In this case the action associated with Step 3 will be performed—after which h will be one letter longer, but still a prefix of s. Now observe that at Step 2, if $h = s$, then none of the queried DFAs will be positive examples, so the action of Step 4 will be correctly performed. Finally note that h is initially a prefix of the target string s, and remains one throughout the execution of the algorithm; thus at Step 2, it is always true that either $h = s$ or h is a proper prefix of s.

Hence this algorithm must converge to the target string. It is also easy to show that it requires at most $|\Sigma| \cdot |s|$ queries. ■

4.2. Determinate arity-two function-free Prolog clauses

Another well-studied problem is the learnability of logic programs (Page & Frisch, 1992, Džeroski et al., 1992, Cohen & Page, 1995). A special case that has received much attention is the learnability of determinate non-recursive function-free bounded-arity one-clause programs. We will denote this language below as a-$\mathcal{D}et\mathcal{LP}$, where a is the bound on arity. Again, we will assume that the reader is familiar with this representation; however Appendix B contains a brief overview of the necessary background on logic programs.

In previous work, Džeroski, Muggleton and Russell (Džeroski et al., 1992) showed that for any constant a, constant depth a-$\mathcal{D}et\mathcal{LP}$ programs are pac-learnable. Later, Kietz (Kietz, 1993) showed that 2-$\mathcal{D}et\mathcal{LP}$ programs of arbitrary depth are not pac-learnable, and Cohen (Cohen, 1993) showed that 3-$\mathcal{D}et\mathcal{LP}$ log-depth programs are not polynomially predictable. To date, however, the polynomial predictability of arbitrary-depth 2-$\mathcal{D}et\mathcal{LP}$ programs has remained an open question.

Kietz considered a learning model similar to the one considered by Cohen and Hirsh: the target concept is a 2-$\mathcal{D}et\mathcal{LP}$ non-recursive function-free clause, and the examples are non-recursive ground clauses which are labeled as positive iff they are *θ-subsumed* (see Appendix B for a definition) by the target clause. We have the following result for this model.

COROLLARY 1 *Under the cryptographic assumptions of Theorem 2, 2-$\mathcal{D}et\mathcal{LP}$ non-recursive function-free clauses are not polynomially predictable in the model of Kietz.*

Proof: Kietz showed that there are functions f_K and g_G such that the following hold: f_K maps a DFA M to a ground clause; g_K maps a string s to a function-free clause; and $s \in L(M)$ iff $g_K(s)$ θ-subsumes $f_K(M)$ (Kietz, 1993, Lemma 13).

Briefly, f_K and g_G are as follows. To compute $f_K(M)$, generate one unary predicate symbol *acc*, one binary predicate symbol p_a for each letter a in Σ and a constant c_{q_i} for each state q_i of M. $f_K(M)$ is the clause with the head *dfa*(c_{q_0}), and a body containing one literal of the form $p_a(q_i, q_j)$ for every transition $\delta(q_i, a) = q_j$ in M and one literal of the form $acc(q_k)$ for every accepting state q_k. To compute $g_K(a_1 \ldots a_n)$, generate $n + 1$ variables $X_0, \ldots, X_n$. Then $g_K(a_1 \ldots a_n)$ is the clause

$$dfa(X_0)\text{:-}p_{a_1}(X_0, X_1), \ldots, p_{a_n}(X_{n-1}, X_n), acc(X_n)$$

Kietz used this construction to reduce the DFA intersection problem to a consistency problem, thus demonstrating that 2-$\mathcal{D}et\mathcal{LP}$ non-recursive function-free clauses are not pac-learnable. However, the construction also constitutes a prediction-preserving reduction from the dual DFA problem to the language of 2-$\mathcal{D}et\mathcal{LP}$ non-recursive function-free clauses.

■

5. Automatic programming and related problems

Another well-studied problem in artificial intelligence is automatic programming from examples (*e.g.,* (Summers, 1977, Biermann, 1978).) Here the goal is to learn a program, typically in a functional language such as LISP, from examples of the form (x, y) where x is an input to the target program and y is the associated output.

Below we will formalize a simple version of this problem. We will consider learning a program that is the composition of n functions from a designated set. This relatively "easy" case is the functional equivalent of n lines of "straight-line" code—code that includes no loops or branches, and additionally operates on a single variable. As a final restriction, we will require that the output of the target function be binary by requiring the target function to be of the form

$$\lambda(x).p(o_n(o_{n-1}(\cdots o_1(x)\cdots)))$$

where the range of $p(x)$ is $\{0, 1\}$. (The other functions $o_1,\ldots,o_n$ will map strings to strings.)

Later, we will wish to augment our formalism to model variants of this problem in which program traces are also available to the learner. First, however, we will define more precisely the problem of learning straight-line code from input/output pairs, as introduced informally above.

5.1. *A formalization of learning straight-line code*

Define an *operator set* $\langle O, P\rangle$ to be a two sets of functions $O = \{o_1, \ldots, o_{k_O}\}$ and $P = \{p_1, \ldots, p_{k_P}\}$ such that each o_i is a function from $\{0, 1\}^{n_I}$ to $\{0, 1\}^{n_I}$ and each p_i is a function from $\{0, 1\}^{n_I}$ to $\{0, 1\}$. The number n_I is the *domain width* of the operator set, and $k_O + k_P$ is its *cardinality*. We will use $\mathcal{OP}_{n_I,n_O}$ to denote the set of all operator sets of domain width at most n_I and cardinality at most n_O. For any operator set $\langle O, P\rangle$, we define $\mathcal{AP}[\langle O, P\rangle]$ be the set of all concepts with characteristic functions of the form[6]

$$o_{i_1} \circ \ldots \circ o_{i_n} \circ p_k$$

where each o_{i_j} is in O and p_k is in P. We will use $\mathcal{AP}[\mathcal{OP}_{n_I,n_O}]$ to denote the set $\{\mathcal{AP}[\langle O, P\rangle] : \langle O, P\rangle \in \mathcal{OP}_{n_I,n_O}\}$. Notice that this is a set of languages, parameterized by the operators O and predicates P.

The size of a concept $C = o_{i_1} \circ \ldots \circ o_{i_n} \circ p_k$ in $\mathcal{AP}[\langle O, P\rangle]$ is defined to be n, the number of operators used.

To motivate the slightly awkward notation introduced above, notice that for some operator sets $\langle O, P\rangle$, the class $\mathcal{AP}[\langle O, P\rangle]$ is easily learnable—for example, if every o_i is the identity function then the problem is trivial. However, one would really like to ask questions about the existence of "operator-independent" learning algorithms—algorithms that work for any given sets O and P. With the definitions given above, we can formalize this question succinctly as follows:

> Is there an algorithm PacPredict that polynomially predicts any concept class $\mathcal{AP}[\langle O, P\rangle] \in \mathcal{AP}[\mathcal{OP}_{n_I,n_O}]$, given O, P, and access to oracles for the functions in O and P ?

If the answer to this question is affirmative then we will say that $\mathcal{AP}[\mathcal{OP}_{n_I,n_O}]$ is *uniformly predictable*. *Uniform pac-learnability* is defined analogously. Thus we have reduced the question of operator-independent learnability to a set of (closely related) questions, posed in the usual pac-learning model.[7]

5.2. *Automatic programming from input-output pairs*

Our first result pertains to automatic programming from input-output pairs. The result shows that even with a small fixed set of operators, and even in the case of straight-line code, this problem can be hard. This result is a little surprising, given that most of the research issues arising in automatic programming seem to relate to problems such as inducing loops and conditional statements, not to difficulties associated with learning straight-line code.

Theorem 5 *For $n_O \geq 4$, $\mathcal{AP}[\mathcal{OP}_{n_I,n_O}]$ is not uniformly predictable, under the cryptographic assumptions of Theorem 2.*

Proof: We will construct a particular operator set $\langle O, P\rangle$ that is hard to predict.

Theorem 2 gives a prediction-preserving reduction from circuits of depth $\log n$ to

$$\mathcal{S}^{\mathrm{DFA}}_{p_I(n),p_T(n)}$$

via two functions $g^{\log n}$ and $f^{\log n}$, where $g^{\log n}$ maps boolean formulae to strings, $f^{\log n}$ maps assignments to DFAs, and p_I and p_T are both polynomials in n. We will use this reduction to reduce circuits to $\mathcal{AP}[\langle O, P\rangle]$ for a specific $\langle O, P\rangle$ pair with $n_O = 4$ and n_I polynomial in n, thus showing that this $\mathcal{AP}[\langle O, P\rangle]$ is not predictable.

Let $n_I = n + \log p_I(n)$, and choose some systematic way of numbering the states in a DFA such that the start state has number 0 (*e.g.*, breadth first). In this way, if η is an assignment to n variables, n_I bits can be used to encode a state in the DFA $f^{\log n}(\eta)$. We will write the encoding of the i-th state in $f^{\log n}(\eta)$ as $c(\eta)c(i)$ where $c(\eta)$ is the encoding of η and $c(i)$ is the encoding of i. We can now define the instance mapping of the new reduction as $f_{\mathrm{AP}}(\eta) \equiv c(\eta)c(0)$.

Now let $O = \{o_0, o_1, o_\star\}$ and $P = \{p_{acc}\}$, where the functions p_{acc}, o_0, o_1, and $o_\star$ are defined as follows. For $a \in \{0, 1, \star\}$, define $o_a(c(\eta)c(i)) \equiv c(\eta)c(j)$ where j is the index of the state in $f^{\log n}(\eta)$ that is reached by traversing the edge labeled a from state i. Also define $p_{acc}(c(\eta)c(i))$ to be 1 if the i-th state of $f^{\log n}(\eta)$ is accepting and 0 if the i-th state of $f^{\log n}(\eta)$ is rejecting. For a formula $b \in \mathcal{B}^d_{n,*}$, we define the concept mapping

$$g_{\mathrm{AP}}(b) \equiv o_{a_1} \circ \ldots \circ o_{a_n} \circ p_{acc}$$

where $g^{\log n}(b) = a_1 \ldots a_n$.

Notice that these particular operators O and P correspond very closely to the arcs in the automata that are the range of $f^{\log n}$. In particular, if $c = c(\eta)c(i)$ encodes a state q_i in an automaton $M_\eta = f^{\log n}(\eta)$, then $o_1(c)$ is the encoding of the state in M_η reached by following the arc labeled 1, $o_0(c)$ is the encoding of the state in M_η reached by following the arc labeled 0, and $o_\star(c)$ is the encoding of the state in M_η reached by following the arc labeled $\star$. Also, the predicate p_{acc} maps exactly those strings that encode accepting states of M_η to 1, and the rejecting states to 0. This means that the string $a_1 \ldots a_n$ is accepted by $f^{\log n}(\eta)$ iff

$$(o_{a_1} \circ \ldots \circ o_{a_n} \circ p_{acc})(c(\eta)c(0)) = 1$$

This property, together with the arguments of Theorem 2, shows that the reduction given above preserves membership as required by the definition of prediction-preserving reducibility.

The remainder of arguments needed to prove the theorem are straightforward. Since $f^{\log n}$ can be computed in polynomial time then so can p_{acc}, o_0, o_1, and $o_\star$, so it is possible to simulate the oracles for O and P. (Recall that $f^{\log n}$ does not depend on the target concept.) Thus if there were a prediction algorithm for $\mathcal{AP}[\langle O, P\rangle]$, one could use it to find an ϵ-good predictor for any log-depth circuit. ■

As another possible proof method, the arguments used in Section 6 of Frazier and Pitt (Frazier & Pitt, 1996) (which shows the hardness of learning CLASSIC from individuals) could also be easily adapted to show that $\mathcal{AP}[\mathcal{OP}_{n_I,n_O}]$ is not uniformly predictable; in fact, their construction can be adapted to show that there exists an $\langle O, P\rangle$ pair such that predicting polynomial-sized circuits is reducible to predicting $\mathcal{AP}[\langle O, P\rangle]$. This result is again incomparable to ours. It is stronger in that it is based on a reduction from polynomial-size circuits, rather than log-depth circuits, so weaker cryptographic assumptions can be made; however, the construction requires an operator set of size polynomial in the size of the target circuit, whereas in our construction the operator set contains a fixed number of operators and predicates (three and one, respectively).

There is also no obvious way to extend the Frazier and Pitt result to the problem of learning from partial traces discussed below.

5.3. Programming by demonstration

Another variant of the automatic programming problem is the problem of learning programs from traces. Here the assumption is that the learner can observe (perhaps only partially) the actions taken by a working copy of the target program.

Programming from traces is closely related to the currently active research area of *programming by demonstration* (Cypher, 1993). Here the operations used in the target program are (typically) operations on a graphical user interface, and the "teacher" from whom traces are obtained is a human user.

In a formal learning model it is natural to model these traces with an oracle. However, in a programming by demonstration context, what a "trace" is depends on the system being used. We will thus consider different oracles, corresponding to different assumptions about what is observable to a programming by demonstration system.

5.3.1. Observable operators. One plausible assumption is that one can observe the sequence of operators $a_1, \ldots, a_n$ used by the teacher. (For instance, a trace might be a sequence of EMACS keystrokes, each of which is an operator.) In the situation considered above, where the target "program" consists of straight-line code, this information is sufficient to identify the target concept. This simple but useful learning mechanism, called a "macro recorder", is well-understood and embedded in many existing user interfaces. We will not consider it further in this paper.

5.3.2. Observable intermediate values. Another reasonable assumption is that one can observe the intermediate values derived by the teacher but not the operators. This might be the case if the operators appearing in the learned program are *abstractions* of the actions actually taken by the user.

For example, consider the problem of learning a sequence of commands that converts an entry in a mailing list from one format to another. The user's actions might be primitive editor commands, such as *forward-one-character*, *back-one-word*, *insert-newline*, and so on. However, the mailing list conversion problem might be more appropriately modeled with abstract operators such as *forward-to-the-beginning-of-sentence*, *back-to-beginning-of-postal-code*, and so on. Designing such abstract operator sets and learning "programs" containing such abstract operators is an important emphasis of at least some programming by demonstration research efforts. One difficulty in learning programs with abstract operators is that the abstract operators are not observable. In learning an editor macro, for instance, one might be able to observe that a user moved the cursor from point p_1 to point p_2 in a document, but it might not be obvious whether this action corresponds to the operator *forward-to-end-of-line* or to the operator *forward-to-end-of-postal-code*.

One can model this situation with an oracle TRACE(x) which, when called with the input x while learning the target concept

$$o_{i_1} \circ \ldots \circ o_{i_n} \circ p_k$$

returns a sequence of strings $\langle s_1, \ldots, s_n \rangle$ where s_j is the j-th intermediate value constructed by the target function. (That is, $s_j = (o_{i_1} \circ \ldots \circ o_{i_j})(x)$.) Let us now consider the problem of learning from examples and the TRACE oracle. While this problem is certainly more complex than recording a macro, it is easy to show that the TRACE oracle is enough to allow efficient learnability in the pac sense.

THEOREM 6 *$\mathcal{AP}[\mathcal{OP}_{n_I,n_O}]$ is uniformly pac-learnable from examples and the TRACE oracle. Furthermore, the prediction algorithm runs in time polynomial in n_O.*[8]

Proof:

Let $\langle O, P\rangle$ be in $\mathcal{OP}_{n_I,n_O}$. We note that the cardinality of $\{C \in \mathcal{AP}[\langle O, P\rangle] : \|C\| < n_T\}$ is bounded by $n_O{}^{(n_T+1)}$, and hence the VC-dimension (Blumer et al., 1989) of $\mathcal{AP}[\langle O, P\rangle]$ is bounded by $(n_T + 1)\log n_O$. To complete the proof we need only show that one can find a hypothesis consistent with any m examples in time polynomial in m, n_I, n_T, and n_O.

Let $x_1, \ldots, x_m$ be the examples. First, invoke the TRACE oracle for each example x_i to get the sequence $\langle s_{i1}, \ldots, s_{in}\rangle$. Now, for each $j : 1 \leq j \leq n$, look for some $o_j \in O$ such that $\forall i : 1 \leq i \leq m$, $o_j(s_{i,j-1}) = s_{i,j}$. This can be done with a linear search through O with mn_On_T calls to the oracle for O. Also look for some $p_j \in P$ such that $\forall i : 1 \leq i \leq m$, $p_k(s_{in})$ agrees with the label of x_i—*i.e.*, $p_k(s_{in}) = 1$ if x_i is a positive examples and $p_k(s_{in}) = 0$ if x_i is a negative example. Again, this can be done with a linear search with mn_O calls to the oracle for G. The hypothesis

$$o_1 \circ \ldots \circ o_n \circ p_k$$

will be consistent with the examples. ■

In passing we note that learnability can also be shown in the exact identification model of learning from equivalence queries—a model which is perhaps more appropriate in this setting.

5.3.3. Partially observable intermediate values. In some circumstances, it may not be appropriate to assume that one has access to the intermediate values computed by the target function. A slightly weaker assumption is to assume that some but not all aspects of the intermediate states are observable. This assumption is perhaps representative of the case in which the intermediate states include a "user goal" which cannot be observed by the learner. Another plausible case in which this assumption might be appropriate is if the state of the system being acted on by the user is too large to transmit to the learner, and hence for efficiency reasons only a partial description of these states is available; for instance, it might be impractical to transmit to the learner the entire state of an editor.

These situations can be modeled as follows. If $s \in \{0, 1\}^n$, then let HIDE$_k(s)$ denote the set of all strings $s' \in \{0, 1, ?\}^n$ that can be derived by changing up to k elements of s from "0" or "1" to "?". For example,

$$\text{HIDE}_1(1101) = \{?101, 1?01, 11?1, 110?, 1101\}$$

We now define the oracle PTRACE$_k(x)$ to return some sequence $\langle h_1, \ldots, h_n\rangle$ such that for all $j : 1 \leq j \leq n$, $h_j \in \text{HIDE}_k((o_{i_1} \circ \ldots \circ o_{i_j})(x))$. In other words, the oracle is allowed to edit the correct intermediate values by changing up to k bits of each value to the symbol ?. Learning from a PTRACE oracle and examples is one possible formalization of learning from partially observable intermediate values.

Notice that while we are assuming that the elements in the PTRACE result from "hiding" at most k bits of the intermediate values, we impose no constraints on which k bits are "hidden". In effect our model assumes that these bits are hidden by an adversary. However, in the constructions below the adversary follows a very simple strategy—the bits hidden are always the lowest order bits of a string.

Recall that in the model of learning from input/output pairs, the trace is completely hidden; thus Theorem 5 shows that if all bits in a trace are hidden, then learning can be hard. (*I.e.*, for $n_O \geq 4$, uniformly predicting $\mathcal{AP}[\mathcal{OP}_{n_I,n_O}]$ from examples and PTRACE$_{n_I}$ is difficult.) This result can be strengthened to show that hiding even a few bits of each intermediate value makes learning hard.

THEOREM 7 *For $n_O \geq 4$ and $k \geq \log(\log n_I)$, $\mathcal{AP}[\mathcal{OP}_{n_I,n_O}]$ is not uniformly predictable from examples and the PTRACE$_k$ oracle, under the cryptographic assumptions of Theorem 2.*

Proof: The argument follows the argument of Theorem 5, in which $\log n$-depth circuits are reduced to a learning problem in $\mathcal{AP}[\mathcal{OP}_{n_I,n_O}]$. Recall that in this reduction, an assignment η to a set of n boolean variables was first mapped the DFA $f^{\log n}\eta$ (which is a leveled DFA of width $\log n$) and then to an encoding of the initial state of this DFA. The operator and predicate sets O and P encoded the transitions within DFAs in the range of $f^{\log n}$.

In this theorem we will use an analogous technique; however, we will encode a state q in $f^{\log n}(\eta)$ in three sections: first, an encoding of η, which requires $O(n)$ bits; second, an encoding of the depth d of the state, which requires $O(\log n)$ bits; and finally, an encoding of where q lies in the d-th level of $f^{\log n}(\eta)$, which requires $O(\log \log n)$ bits.

The arguments of Theorem 5 show how examples are converted, and also how the oracles for O and P can be simulated. To prove the theorem it is only necessary to show how the PTRACE oracle can be emulated.

To emulate the oracle for PTRACE(x) on $x = f^{\log n}(\eta)$, simply construct $f^{\log n}(\eta)$ and choose a random path to an accept state (if x is positive) or a reject state (if x is negative). Representing these states with the encoding above gives a sequence of intermediate values $s_1, \ldots, s_n$. Of course, these need not agree with the actual values $s_1^*, \ldots, s_n^*$ taken by the automata in accepting the target string; however, they will agree in all bits *except* those $O(\log \log n)$ bits needed to encode the position within a level. To construct a legal output for PTRACE, therefore, it is only necessary to "hide" these bits by replacing them with ?. Thus if exactly these bits are "hidden" the resulting sequence $h_1, \ldots, h_n$ is a legal output for PTRACE. ■

This theorem shows that allowing an adversary to hide even a tiny part of each intermediate value in a trace makes learning difficult. Carrying this line of investigation a little further, one can show that even hiding a small *constant* number of bits in each element of the trace can make learning as hard as learning boolean functions in disjunctive normal form (DNF), an open problem in computational learning theory.

THEOREM 8 *For $n_O \geq 4$ and $k \geq 2$, $\mathcal{AP}[\mathcal{OP}_{n_I,n_O}]$ is not uniformly predictable from examples and the PTRACE$_k$ oracle unless DNF is polynomially predictable.*

Proof: In the proof of Theorem 7, the key insight is that the dual DFA problem (for leveled width w automata) can be reduced to the problem of automatic programming from traces, where three operators o_j and one predicate p_k are available, and the number of bits hidden by PTRACE is logarithmic in w, the maximal automata width. In other words, predicting $\mathcal{S}_{n_I,*}^{\mathrm{LDFA}(w)}$ can be reduced to a prediction problem $\mathcal{AP}[\langle O, P\rangle] \in \mathcal{AP}[\mathcal{OP}_{poly(n_I),n_O=4}]$,

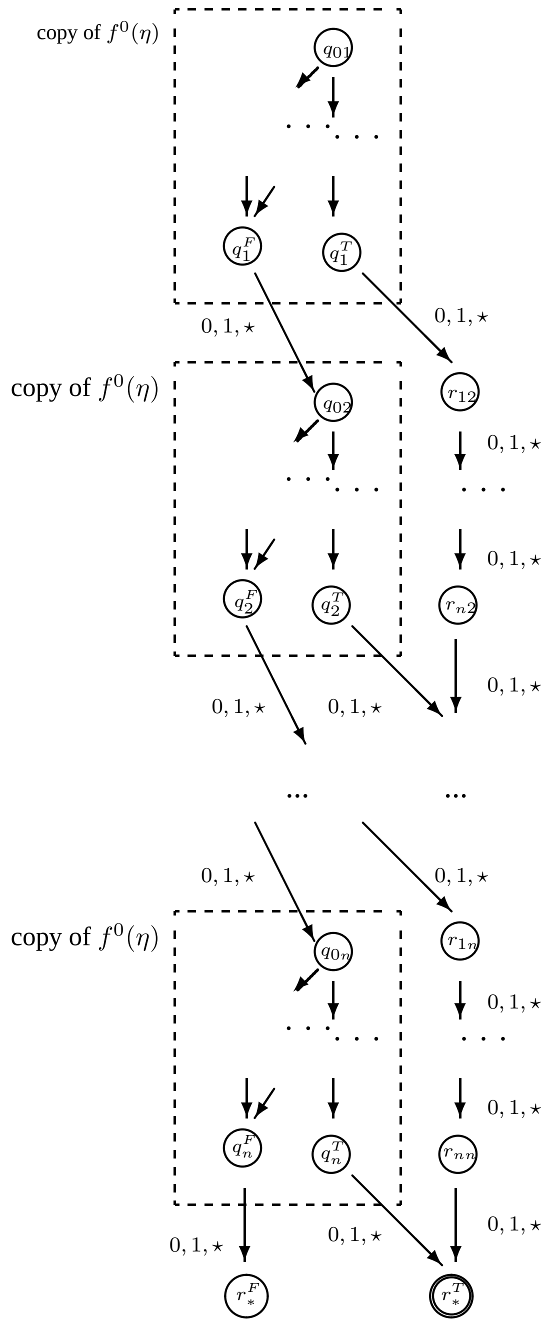


Figure 4. The instance mapping for the reduction from DNF to $\mathcal{S}^{\mathrm{LDFA}(3)}$

where the learning is done from examples and a PTRACE$_{\log w}$ oracle. Thus to prove the theorem it is only necessary to show a prediction-preserving reducibility from DNF over n variables to $\mathcal{S}^{\mathrm{LDFA}(4)}_{poly(n),*}$, the dual DFA problem for leveled automata with level width 4 or less. We will describe this reduction below.

Let $T_1, \ldots, T_s$ be monomials, and let $\phi = \vee_{i=1}^{s} T_i$ be an polynomial-sized DNF formula of size n_T over n variables. Assume without loss of generality that the number of terms s is equal to the number of variables n. (By padding we can make these quantities equal). We define the concept mapping g_{DNF} to be

$$g_{\mathrm{DNF}}(T_1 \vee T_2 \vee \ldots \vee T_n) = g^0(T_1) \star g^0(T_2) \star \ \ldots \ \star g^0(T_n) \ \star$$

where $g^0 : \mathcal{B}^0_{n,*} \rightarrow \{0, 1, \star\}^n$ is the concept mapping used in Theorem 2. For example, if $n = 4$,

$$g_{\mathrm{DNF}}(x_1\overline{x_2} \vee x_2\overline{x_3} \vee x_3\overline{x_4}) = \underline{10\star\star} \star \underline{\star 10 \star} \star \underline{\star\star 10} \star$$

Recall that $f^0(\eta)$ maps an assignment η to a DFA that accepts exactly those strings that encode monomials satisfied by η. For an assignment η, we define the instance mapping $f_{\mathrm{DNF}}(\eta)$ to consist of n copies of $f^0(\eta)$, together with a linear sequence of "success" states. These copies will be connected so that if the automata reaches the accept state of any copy of $f^0(\eta)$, it will always jump to a "success" state; however if the automata reaches the reject state of a copy of $f^0(\eta)$, it will always proceed to the start state of the next copy. The automata accepts if it eventually reaches the "success" sequence. Such an automaton is shown in Figure 4. Below we will define it more formally.

- The states of $f_{\mathrm{DNF}}(\eta)$ include n copies of each state q in the state set of $f^0(\eta)$, where $f^0(\eta)$ is the instance mapping used in Theorem 2.

  We will denote the i-th copy of the start state q_0 as q_{0i}, the i-th copy of the accepting state q_n^T as q_i^T, the i-th copy of the maximal-depth rejecting state q_n^F as q_i^F, and the i-th copy of an arbitrary state q as q_i.

- If q and q' are connected by an arc labeled a in $f^0(\eta)$, then for $i : 1 \leq i \leq n$, q_i and q'_i are also connected by an arc labeled a in $f_{\mathrm{DNF}}(\eta)$. These arcs complete the n copies of $f^0(\eta)$.

- The states of $f_{\mathrm{DNF}}(\eta)$ also include $n(n-1)$ states named $r_{12}, \ldots, r_{n2}, \ldots, r_{1n}, \ldots, r_{nn}$, and two additional states r_*^T and r_*^F. (These are the states of the linear sequence of "success states" mentioned above.)

- For $i : 2 \leq i < n$, there are arcs labeled 0, 1, and $\star$ from $r_{n,i}$ to $r_{1,i+1}$. These arcs complete the "success sequence".

- The state q_{01} is the start state, and the state r_*^T is the sole accepting state.

- For $i : 1 \leq i < n$, there are arcs labeled 0, 1, and $\star$ from q_i^T to $r_{1,i+1}$, and arcs labeled 0, 1, and $\star$ from q_i^F to $q_{0,i+1}$. Thus if the i-th copy of $f^0(\eta)$ succeeds, the automaton will jump to the "success sequence" of r_i's; if the i-th copy fails, then the automaton will go on to the next copy.
- There are arcs labeled 0, 1, and $\star$ from $r_{n,n}$ to r_*^T, arcs labeled 0, 1, and $\star$ from q_n^T to r_*^T, and arcs labeled 0, 1, and $\star$ from q_n^F to r_*^F. Thus if any of the first $n-1$ copies succeed, the automaton will jump to the correct point in the "success sequence"; further the automaton will succeed if the success sequence is reached, or if the final copy succeeds.

Clearly the size of this construction is polynomial, it can be computed in polynomial time, and the level width of the DFA is bounded by 3. The arguments for the correctness of the mapping parallel the arguments used in Theorem 2. ■

To summarize the results of this section, we have investigated an extension of the problem of recording a macro, in which the goal is to learn a linear sequence of operators taken from a known set. We showed that this learning problem is trivial if the operators are observable, and tractable if the operators are hidden and the intermediate states of the computation are observable. However, if the intermediate states are hidden, then the problem becomes intractable. More surprisingly, the problem is hard even if the intermediate states are only partially hidden—hiding even two bits of each intermediate value makes learning as hard as learning DNF, and hiding $O(\log \log n)$ bits makes learning cryptographically hard.

6. Conclusions

In this paper we analyzed a simple instance of a learning problem involving structured examples; specifically, we analyzed a dual version of the problem of learning DFAs, in which examples are DFAs, concepts are strings, and a string denotes the set of DFAs that accept it. The dual DFA learning problem is a formalization of a problem in which concepts are relatively simple, but examples are allowed to have a non-trivial structure: namely, the structure of a rooted directed graph. We showed that the "dual DFA problem" is as hard as learning log-depth boolean circuits, even if example DFAs are restricted to be over a three-letter alphabet and also acyclic, leveled, and of logarithmic level width.

Corollaries of this result answer two open questions in the learnability of first-order representations. First, under cryptographic assumptions, the description logic CLASSIC is not learnable in the model proposed by Cohen and Hirsh (Cohen & Hirsh, 1994a). Second, under cryptographic assumptions, arity-two "determinate" function-free Prolog clauses are not polynomially predictable in the model proposed by Kietz (Kietz, 1993).

The dual DFA result also has implications for the problem of learning straight-line programs (without branches or loops) from input/output pairs or traces—a trace being a sequence revealing the intermediate values required to evaluate the target function on an example x. We motivated a particular formalization of this problem and showed that learning from input/output pairs is cryptographically hard, but that learning from traces is tractable. As an intermediate between these two models, we then proposed a model of learning from *partial traces*. In particular we considered learning from examples and the oracle PTRACE$_k$, which can be thought of as returning a complete trace that has been edited by an adversary who can "hide" at most k bits of every intermediate value. We showed that learning from partial traces is cryptographically hard even if only $O(\log \log n)$ bits of each value are hidden. Furthermore, learning from partial traces is as hard as learning DNF even if only two bits of each value are hidden. These results may have implications for the research area of "programming by demonstration" (Cypher, 1993).

We will conclude with some further remarks on the implications of these results, and more generally, on the role of negative formal results in computer science. The computer science community has three main goals: to identify problems that are worthy of study, to understand these problems, and to engineer solutions to them. The last two goals are often closely related, since better solutions often arise from better understanding. While negative formal results seldom immediately suggest a new engineering solution to a problem, they can and often do lead to progress in our collective understanding of a problem.

With respect to the results of this paper, previous formal results have provided considerable insight into the computational complexity of many types of first-order learning—one exception being the case of learning logic program clauses over binary determinate predicates. This is an important special case for both practical and formal reasons. Practically, it is related to widely used representations such as description logics and functional programming languages. Formally, while the language is known not be properly learnable (Kietz, 1993), recent positive results have shown that some interesting subclasses can be learned using novel representation schemes for hypotheses (Horváth et al., 1997).

The formal results suggest that determinate Prolog clauses may have different learnability properties in the arity-two case than in the more general case, in which predicates may have arity three or more; in particular, it raises the possibility that a large subset of this practically important special case can be efficiently learned, if an appropriate representation for hypotheses is used. If this were the case, it might have important implications for the design of future first-order learning systems (which should arguably be extended to deal appropriately with the special case) as well as future knowledge representations systems (which should arguably be extended to support the representations used as hypotheses of the learners.)

However, the hardness result of this paper gives a strong upper bound on what sort of arity-two clauses can be learned; specifically, it implies that the assumption of binary determinate predicates alone is not enough to guarantee learnability. In addition, the result clarifies our understanding of the problem in several important respects. In particular, the proofs indicate what sort of additional restrictions might lead to further positive learnability results. For instance, to obtain a positive result, it would clearly not be sufficient to restrict the number of available predicates to an arbitrary constant. However, we observe that in the proofs, it is necessary to make an adversarial choice of both the transition function and state labelings of the example DFAs. Interestingly, prior results in DFA learning show that while DFAs are hard to learn given an adversarial choice of target concepts, learning is sometimes possible in only slightly less adversarial settings; as an example, consider the distributions of "typical" DFAs considered by Freund *et. al* (Freund et al., 1993), in which an adversary determines the transition function of the target DFA, but the labeling of states is determined stochastically. We leave as an open question the complexity of the dual DFA learning problem in an analogous setting.

Acknowledgments

I would like to thank Haym Hirsh, Lenny Pitt, and Rob Schapire for a number of helpful comments on this paper.

Appendix A

Semantics of CoreClassic

Below we will briefly review the semantics of CoreClassic, as they pertain to the results of this paper. Readers are referred to Borgida and Patel-Schneider (Borgida & Patel-Schneider, 1994) for a fuller discussion, or to Cohen and Hirsh (Cohen & Hirsh, 1994a, Cohen & Hirsh, 1994b) or Frazier and Pitt (Frazier & Pitt, 1996) for a discussion in the context of learnability problems.

Concepts in CoreClassic describe subsets of a domain I of “individuals”. Concepts are built from a alphabet of *primitive class symbols* p, each of which corresponds to a subset of I; *role symbols* r, each of which corresponds to a subset of $I \times I$; *attribute symbols* a, each of which corresponds to a function from I to I; and the operators AND, ALL, and SAME-AS.

For a primitive p, let $ext(p)$ denote the subset of I that corresponds to p; for a role r, let $r(x, y)$ be the corresponding binary predicate; and for an attribute a, let $a(x)$ be the corresponding function. A CoreClassic concept is defined inductively as follows.

- If p is a primitive class symbol then p is a concept denoting $ext(p)$.
- If r is a role or attribute and C is a concept, then (ALL r C) is a concept, denoting the set of all $x \in I$ such that $\forall y \in I,\ r(x, y) \Rightarrow y \in ext(C)$, where $ext(C)$ is the set denoted by the concept C.
- If $a_1, \ldots, a_k, b_1, \ldots, b_l$ are attribute symbols, then (SAME-AS ($a_1 \ldots a_k$) ($b_1 \ldots b_l$)) is a concept denoting the set of $x \in I$ such that $a_k(\cdots a_2(a_1(x))\cdots) = b_l(\cdots b_2(b_1(x))\cdots)$.
- If $C_1, \ldots, C_n$ are concepts then (AND $C_1 \ldots C_n$) is a concept denoting $\bigcap_{i=1}^{n} ext(C_i)$.

An important relationship in description logics is *subsumption*. Concept C_1 subsumes C_2 if $ext(C_1) \supseteq ext(C_2)$ regardless of the extensions of the primitive concepts, roles and attributes used in C_1 and C_2.

For example the concept $C_1 =$ `(AND politician lawyer)` would not subsume the concept $C_2 =$ `(AND congressman lawyer (SAME-AS mistress aide))` even if it happened to be the case that $ext($`politician`$) \supset ext($`congressman`$)$ and hence $ext(C_1) \supseteq ext(C_2)$. This is because for C_1 to subsume C_2, it must be that $ext(C_1) \supseteq ext(C_2)$ regardless of how the primitive concepts are defined, and it it clearly possible to define the “politician” and “congressman” so that $ext($`politician`$) \not\supset ext($`congressman`$)$. However, $C_1' =$ `(AND congressman lawyer)` would subsume C_2, since every element of C_1' is necessarily a member of C_2, regardless of the definition of the primitive roles and concepts.

Appendix B

Semantics of logic programs

In the interests of simplicity, the definitions below only coincide with the usual ones for the case of non-recursive function-free single-clause Prolog programs. For a more complete description of logic programming see one of the standard texts (*e.g.,* (Lloyd, 1987)).

Logic programs are written over an alphabet of *constant symbols*, *predicate symbols*, and *variables*. A *function-free literal* is written $p(X_1, \ldots, X_k)$ where p is a predicate symbol and $X_1, \ldots, X_k$ are variables. A *fact* is written $p(t_1, \ldots, t_k)$ where p is a predicate symbol and $t_1, \ldots, t_k$ are constant symbols. The number of arguments k to a literal (or fact) is called its *arity*.

A *ground clause* is written a:-$b_1, \ldots, b_n$, where a and the b_i's are all facts. A *function-free clause* is written C:-$D_1, \ldots, D_n$, where C and the D_i's are all function-free literals. The fact (or literal) to the left of the ":-" symbol is the *head* of the clause and the facts (literals) to the right of the ":-" symbol are the *body* of the clause.

A *substitution* is a partial function mapping variables to constant symbols or variables. If θ is a substitution and A is a literal, we will use $A\theta$ to denote the result of replacing each variable X in A with the constant symbol to which X is mapped by θ.

The function-free clause C:-$D_1, \ldots, D_n$ is said to θ*-subsume* the ground clause a:-$b_1, \ldots,$ b_m if there is some substitution θ such that $C\theta = a$ and $\forall i : 1 \leq i \leq n, D_i\theta \in \{b_1, \ldots, b_m\}$.

The following restrictions (which assume the literals in the body of a clause to be ordered) are modified from Muggleton and Feng (Muggleton & Feng, 1992). If C:-$D_1 \wedge \ldots \wedge D_n$ is a function-free clause, then the *input variables* of the literal D_i are those variables appearing in D_i that also appear in the clause C:-$D_1 \wedge \ldots \wedge D_{i-1}$; all other variables appearing in D_i are called *output variables*. A literal D_i is *determinate* (with respect to a ground clause a:-$b_1, \ldots, b_m$) if for every possible substitution σ such that C:-$D_1 \wedge \ldots \wedge D_{i-1}$ θ-subsumes a:-$b_1 \wedge \ldots \wedge b_m$, there is at most one substitution θ so that $D_i\sigma\theta \in \{b_1, \ldots, b_m\}$. Less formally, a literal is determinate if its output variables have only one possible binding—that is, if the predicate associated with the literal denotes a function, rather than an arbitrary relation.

A function-free clause C:-$D_1 \wedge \ldots \wedge D_n$ is *determinate* with respect to a ground clause a:-$b_1, \ldots, b_m$ if every literal D_i in the body of the clause is determinate with respect to the ground clause. If $\mathcal{D}$ is a distribution over ground clauses, a function-free clause is *determinate* (with respect to $\mathcal{D}$) if it is determinate with respect to every ground clause with non-zero weight under $\mathcal{D}$.

Finally, define the *depth* of a variable appearing in a function-free clause C:-$D_1 \wedge \ldots \wedge D_n$ as follows. Variables appearing in the head of a clause have depth zero. Otherwise, let D_i be the first literal containing the variable X, and let d be the maximal depth of the input variables of D_i; then the depth of X is $d + 1$. The depth of a clause is the maximal depth of any variable in the clause.

Notes

1. To clarify this remark, a typical learning problem includes a set of possible concepts $C_1, \ldots, C_n$ and a set of possible instances $x_1, \ldots, x_m$. Conceptually, one can think of these sets as a large 0,1 matrix in which the columns correspond to instances, the rows correspond to concepts, and each row encodes the characteristic function for a concept. For every such learning problem there is a dual learning problem which is obtained by considering the transpose of this matrix.
2. More precisely, the prediction problem is intractable if one or more of the following are intractable: solving the quadratic residue problem, inverting the RSA encryption function, or factoring Blum integers (Kearns & Valiant, 1989).
3. In fact, under the additional cryptographic assumption that solving the $n \times n^{1+\epsilon}$ subset sum is hard, log-depth circuits are hard to pac-predict even if examples are drawn from a uniform distribution (Kharitonov, 1992).
4. Assuming that examples are concepts is equivalent to assuming that examples are represented by detailed (but polynomial-sized) descriptions of themselves in CoreClassic. This "single-representation trick" is formally convenient, as it avoids introducing a second language for describing instances, and is also sometimes used in experimental AI systems (Dietterich et al., 1982).
5. An early "proof" that CoreClassic was hard to pac-predict turned out to be erroneous (Cohen & Hirsh, 1992, Cohen & Hirsh, 1995).
6. In this paper we follow the convention that $(f \circ g)(x) \equiv f(g(x))$.
7. In previous work, we have used a similar formalization to analyze inductive logic programming learnability problems (Cohen, 1995); in these cases, logic languages are parameterized by a set of available "background predicates".
8. Notice that if PacPredict is a uniform prediction algorithm, then PacPredict must run in time polynomial in n_I, since n_I is the size of the examples used by PacPredict. However, PacPredict need *not* run in time polynomial in n_O.

References

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

Angluin, D. (1988). Learning with hints (extended abstract). In *Proceedings of the 1988 Workshop on Computational Learning Theory*, Boston, Massachusetts.

Beck, H., Gala, H., & Navathe, S. (1989). Classification as a query processing technique in the CANDIDE semantic model. In *Proceedings of the Data Engineering Conference*, pages 572–581, Los Angeles, California.

Biermann, A. (1978). The inference of regular lisp programs from examples. *IEEE Transactions on Systems, Man and Cybernetics*, 8(8).

Blumer, A., Ehrenfeucht, A., Haussler, D., & Warmuth, M. (1989). Classifying learnable concepts with the Vapnik-Chervonenkis dimension. *Journal of the Association for Computing Machinery*, 36(4):929–965.

Boppana, R. & Sipser, M. (1990). The complexity of finite functions. In *Handbook of Theoretical Computer Science*, pages 758–804. Elsevier.

Borgida, A. (1992). Description logics are not just for the flightless-birds: a new look at the utility and foundations of description logics. Technical Report DCS-TR-295, Rutgers University Department of Computer Science.

Borgida, A. & Patel-Schneider, P. F. (1994). A semantics and complete algorithm for subsumption in the CLASSIC description logic. *Journal of Artificial Intelligence Research*, 1:277–308.

Cohen, W. W. (1993). Cryptographic limitations on learning one-clause logic programs. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, Washington, D.C.

Cohen, W. W. (1995). Pac-learning recursive logic programs: negative results. *Journal of AI Research*, 2:541–573.

Cohen, W. W. & Hirsh, H. (1992). Learnability of description logics. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, Pittsburgh, Pennsylvania. ACM Press.

Cohen, W. W. & Hirsh, H. (1994a). The learnability of description logics with equality constraints. *Machine Learning*, 17(2/3).

Cohen, W. W. & Hirsh, H. (1994b). Learning the CLASSIC description logic: Theoretical and experimental results. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourth International Conference (KR94)*. Morgan Kaufmann.

Cohen, W. W. & Hirsh, H. (1995). Corrigendum for "learnability of description logics". In *Proceedings of the Eighth Annual Workshop on Computational Learning Theory*, Santa Cruz, California. ACM Press.

Cohen, W. W. & Page, C. D. (1995). Polynomial learnability and inductive logic programming: Methods and results. *New Generation Computing*, 13(3).

Cypher, A., editor (1993). *Watch what I do: Programming by demonstration*. The MIT Press, Cambridge, Massachusetts.

De Raedt, L., editor (1995). *Advances in Inductive Logic Programming*. IOS Press.

Devanbu, P., Brachman, R. J., Selfridge, P., & Ballard, B. (1991). LaSSIE: A knowledge-based software information system. *Communications of the ACM*, 35(5).

Dietterich, T. G., Lathrop, R. H., & Lozano-Perez, T. (1997). Solving the multiple-instance problem with axis-parallel rectangles. *Artificial Intelligence*, 89(1–2):31–71.

Dietterich, T. G., London, B., Clarkson, K., & Dromey, G. (1982). Learning and inductive inference. In Cohen, P. and Feigenbaum, E. A., editors, *The Handbook of Artificial Intelligence, Volume III*. William Kaufmann, Los Altos, CA.

Džeroski, S., Muggleton, S., & Russell, S. (1992). Pac-learnability of determinate logic programs. In *Proceedings of the 1992 Workshop on Computational Learning Theory*, Pittsburgh, Pennsylvania.

Ergün, F., Kumar, S. R., & Rubinfeld, R. (1995). On learning bounded-width branching programs. In *Proceedings of the Eighth Annual ACM Conference on Computational Learning Theory*, Santa Cruz, CA. ACM Press.

Frazier, M. & Pitt, L. (1996). Classic learning. To appear in *Machine Learning*.

Freund, Y., Kearns, M., Ron, D., Rubinfeld, R., Schapire, R., & Sellie, L. (1993). Efficient learning of typical finite automata from random walks. In *Proceedings of the 25th ACM Symposium on the Theory of Computing*, pages 315–324. ACM Press.

Hopcroft, J. E. & Ullman, J. D. (1979). *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley.

Horváth, T., Sloan, R., & Tuŕan, G. (1997). Learning logic programs by using the product homomorphism method. In *Proceedings of the Tenth Annual ACM Conference on Computational Learning Theory*, Vanderbilt, Tennessee. ACM Press.

Kearns, M. & Valiant, L. (1989). Cryptographic limitations on learning Boolean formulae and finite automata. In *21th Annual Symposium on the Theory of Computing*. ACM Press.

Kharitonov, M. (1992). Cryptographic lower bounds on the learnability of boolean functions on the uniform distribution. In *Proceedings of the Fourth Annual Workshop on Computational Learning Theory*, Pittsburgh, Pennsylvania. ACM Press.

Kietz, J.-U. (1993). Some computational lower bounds for the computational complexity of inductive logic programming. In *Proceedings of the 1993 European Conference on Machine Learning*, Vienna, Austria.

Lloyd, J. W. (1987). *Foundations of Logic Programming: Second Edition*. Springer-Verlag.

MacGregor, R. M. (1991). The evolving technology of classification-based knowledge representation systems. In Sowa, J., editor, *Principles of semantic networks: explorations in the representation of knowledge*. Morgan Kaufmann.

Mays, E., Apte, C., Griesmer, J., & Kastner, J. (1987). Organizing knowledge in a complex financial domain. *IEEE Expert*, pages 61–70.

Muggleton, S. & De Raedt, L. (1994). Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19/20(7):629–679.

Muggleton, S. & Feng, C. (1992). Efficient induction of logic programs. In *Inductive Logic Programming*. Academic Press.

Page, C. D. & Frisch, A. M. (1992). Generalization and learnability: A study of constrained atoms. In *Inductive Logic Programming*. Academic Press.

Pitt, L. & Valiant, L. (1988). Computational limitations on learning from examples. *Journal of the ACM*, 35(4):965–984.

Pitt, L. & Warmuth, M. (1990). Prediction-preserving reducibility. *Journal of Computer and System Sciences*, 41:430–467.

Quinlan, J. R. (1990). Learning logical definitions from relations. *Machine Learning*, 5(3).

Summers, P. D. (1977). A methodology for LISP program construction from examples. *Journal of the Association for Computing Machinery*, 24(1):161–175.

Valiant, L. G. (1984). A theory of the learnable. *Communications of the ACM*, 27(11).

Woods, W. A. & Schmolze, J. G. (1992). The KL-ONE family. *Computers And Mathematics With Applications*, 23(2-5).

Wright, J., Weixelbaum, E., Vesonder, G., Brown, K., Palmer, S., Berman, J., & Moore, H. (1993). A knowledge-based configurator that supports sales engineering and manufacturing and AT&T network systems. *AI Magazine*, 14:69–80.

Received October 7, 1996
Accepted June 12, 1997
Final Manuscript July 3, 1997