# On Learning Read-$k$-Satisfy-$j$ DNF

**Avrim Blum**[*] Carnegie Mellon University Pittsburgh, PA 15213

**Roni Khardon**[†] Harvard University Cambridge, MA 02138

**Eyal Kushilevitz**[‡] Technion Haifa, Israel 32000

**Leonard Pitt**[§] University of Illinois Urbana, IL 61801

**Dan Roth**[¶] Harvard University Cambridge, MA 02138

## Abstract

We study the learnability of Read-$k$-Satisfy-$j$ (RkSj) DNF formulae. These are DNF formulae in which the maximal number of occurrences of a variable is bounded by $k$, and the number of terms satisfied by any assignment is at most $j$. We show that this class of functions is learnable in polynomial time, using Equivalence and Membership Queries, as long as $k \cdot j = O(\frac{\log n}{\log \log n})$. Learnability was previously known only in case that both $k$ and $j$ are constants. We also present a family of boolean functions that have short ($poly(n)$) Read-2-Satisfy-1 DNF formulae but require CNF formulae of size $2^{\Omega(\sqrt{n})}$. Therefore, our result does not seem to follow from the recent learnability result of [Bsh93].

[*]e-mail: avrim@cs.cmu.edu. Research supported in part by an NSF Postdoctoral Fellowship and NSF National Young Investigator grant CCR-93-57793.

[†]e-mail: roni@das.harvard.edu. Research supported by grant DAAL03-92-G-0164 (Center for Intelligent Control Systems).

[‡]e-mail: eyalk@cs.technion.ac.il. Part of this research was done while the author was at Aiken Computation Laboratory, Harvard University, supported by research contracts ONR-N0001491-J-1981 and NSF-CCR-90-07677.

[§]e-mail: pitt@cs.uiuc.edu. Research supported in part by NSF grant IRI-9014840.

[¶]e-mail: danr@das.harvard.edu. Research supported by NSF grant CCR-92-00884 and by DARPA AFOSR-F4962-92-J-0466.

## 1 Introduction

The learnability of DNF formulae has been extensively studied since the seminal paper of Valiant [Val84]. In the distribution free (PAC) model, the only positive results were obtained for DNF formulae with a constant number of terms, DNF formulae with a constant number of literals in a term, or in general subsets that are "polynomially explainable" [KR93]. Cryptographic limitations show [AK91] that for general DNF formulae, in some sense, membership queries cannot help: learnability with membership queries implies learnability without membership queries (under polynomially bounded distributions). Nevertheless, many subclasses of DNF seem to escape the cryptographic constructions in [AK91]. For those subclasses, there are positive learning results when membership queries are allowed, but not without membership queries. Among the learnable subclasses are: Monotone DNF [Ang88, Val84], Read twice DNF [Han91, AP91, PR93], "Horn" DNF (at most one literal negated in every term) [AFP92], $\log n$ term DNF [BR92], Read-$k$-Satisfy-$j$ DNF for constants $k$ and $j$ [AP92] and DNF $\cap$ CNF (this class includes decision trees) [Bsh93]. For many of those subclasses, it is easy to see that learning them (without membership queries) in the PAC model implies learning general DNF in the PAC model.

Following [AP92], we study the learnability of boolean functions parameterized by their Read-$k$-Satisfy-$j$ DNF representation (RkSj, for short). An RkSj DNF representation for a function is a representation in which the maximal number of occurrences of a variable is bounded by $k$ and each assignment satisfies at most $j$ terms in the representation. Clearly, Read-$k$-Satisfy-$k$ DNF is a generalization of $k$-term DNF. The special case of RkSj for $j = 1$ is sometimes referred to as *disjoint* DNF, and we denote it as RkD.

Some results concerning the learnability of disjoint, and almost-disjoint DNF have been obtained in previous work. First, we note that one can extend the results of Kushilevitz and Mansour [KM93] on decision trees to show exact learnability of $\log n$ Disjoint DNF (i.e., Disjoint DNF where all the terms are of length at most $\log n$), and of Disjoint DNF under the uniform distribu-

tion [Kha94]. Second, some recent hardness results for learning DNF in restricted models apply for Disjoint DNF as well: Blum et al. [BFJ+94] prove a hardness result for learning $\log n$ Disjoint DNF in the Statistical Queries model (note that in this model the learner does not have an access to a membership oracle). Aizenstein and Pitt [AP93] show that even if subset queries are allowed, Disjoint DNF is not learnable by algorithms that collect prime implicants in a greedy manner. Further, Aizenstein and Pitt [AP92] prove the learnability of RkSj DNF expressions, for constant $k$ and $j$.

In this paper, we describe certain properties of RkSj formulae which are useful for learnability. We then develop a learning algorithm that uses these properties, in addition to some others found by [AP92], to do a kind of a probabilistic search in the spirit of [BR92]. We show the learnability of RkSj DNF for $k \cdot j = O(\frac{\log n}{\log \log n})$. The learning algorithm uses a typical Equivalence and Membership queries strategy. On a positive counterexample the algorithm (with high probability) adds to its hypothesis a polynomial number of terms one of which is a term in the representation. This improves over the algorithm of [AP92] that adds $n^{O(kj)}$ terms and therefore is polynomial only for constant $k$ and $j$ (although the algorithm of [AP92] is deterministic, and the one presented here is probabilistic). On a negative counterexample the algorithm deletes terms from the hypothesis that are satisfied by that example (and therefore are not implicants of the function). The number of queries is bounded by a polynomial in $n$ and the number of terms in the representation and this implies the learnability.

Clearly, every boolean function has a disjoint DNF expression (simply take a fundamental product of exactly $n$ literals for each satisfying assignment), but a function with a short DNF representation might have an exponentially long disjoint representation. We note that the class of polynomial-size Disjoint DNF contains the class of polynomial-size decision trees: just construct a term for every positive leaf in the tree. On the other hand, we show that Read-2 Disjoint DNF formulae do not have small CNF representations (and therefore do not have small decision trees). In particular, we present a family of functions $\{F_n\}$ that have short ($poly(n)$) Read-2 disjoint DNF formulae but require CNF formulae of size $2^{\Omega(\sqrt{n})}$. (Implying also that the size of a decision tree for the function is super-polynomial.) Therefore our result is not implied, at least in any obvious way, by the recent learnability result for formulae with both polynomial-size CNF and polynomial-size DNF [Bsh93].

In addition to extending the RkSj results of [AP92], our results can be thought of as an extension of previous results for learning $k$-term DNF formulae [BR92, Bsh93], though a true generalization of those results would learn RkSj DNF for $k + j = O(\log n)$. We leave the latter as an interesting open problem.

The rest of the paper is organized as follows: Section 2 includes preliminary definitions and a description of the learning model. In Section 3 we prove some useful properties of RkSj DNF formulae. In Section 4 we present the learning algorithm and prove its correctness. In Section 5 we show that R2D DNF does not have small CNF representation (and in particular no small decision trees).

## 2 Preliminaries

Let $X$ be a set of $n$ variables $x_1, \ldots, x_n$ and define a *literal* to be either a variable or its negation. An *example* (or *assignment*) $a$ is a boolean vector over $\{0,1\}^n$, and we write $a[i]$ to denote the $i$th bit of $a$. A *term* is a conjunction of literals. An example $a$ satisfies a term $t$ (denoted by $t(a) = 1$) if and only if it satisfies all the literals in $t$. For a set of literals $w$, let $term(w)$ be the conjunction of all elements in $w$.

For two boolean functions $F_1, F_2$ we write that $F_1$ implies $F_2$ to mean that, for every assignment $a$, if $F_1(a) = 1$ then $F_2(a) = 1$. A term $t$ is an *implicant* of a DNF formula $F$ if $t$ implies $F$; it is a *prime implicant* if no proper subset of $t$ is also an implicant of $F$.

We often want to look at examples obtained by changing the value of a given example on a specified literal. Toward this end, we introduce the following notation: let $a \in \{0,1\}^n$ be an example, $\ell$ a literal, and $i$ an index. We define the example $b = \mathit{flip}(a, i)$ by: $b[i] = 1 - a[i]$ and $b[i'] = a[i']$ for $i' \neq i$. For example if $a = 1101$ then $\mathit{flip}(a, 3) = 1111$. We also define $\mathit{flip}(a, \ell)$ to be the example $\mathit{flip}(a, i)$ for $i$ such that $\ell \in \{x_i, \overline{x_i}\}$.

The learning model we use is the standard equivalence and membership queries model [Ang88]. In this model, a learner may either propose a hypothesis $H$ (an *equivalence query*) and receive a counterexample $a$ such that $F(a) \neq H(a)$ if one exists, or else propose an example (a *membership query*) and receive the value of $F$ on that example. We say that the learner *learns* a class of functions, if for every function $F$ in the class, the learner succeeds with high probability in finding a hypothesis $H$ logically equivalent to $F$. We allow the learner to use as its hypothesis for the equivalence query any DNF formula. Learnability in this model also implies learnability in the PAC model with membership queries [Ang88].

## 3 Properties of RkSj DNF formulae

Let $F = t_1 \vee t_2 \vee \ldots \vee t_s$ be an RkSj formula. Let $t$ be a term in $F$ and $a$ a satisfying assignment of $t$. The term $t$ has two types of literals in it with respect to $a$. The first type are literals that $F$ is sensitive to, i.e., changing the assignment of $a$ on any of these literals changes the value of $F$. The other type are the literals that do not have this property with respect to the example $a$. More formally, we define:

**Definition 1** *For an assignment $a \in \{0,1\}^n$ and a function $F$ we define the set of* sensitive *literals by*

$$\mathit{sensitive}(a) = \{\ell \mid F(\mathit{flip}(a, \ell)) \neq F(a)\}.$$

A term is *almost satisfied* by an assignment $a$ with respect to $i$ if $t(a) = 0$ but $t(\mathit{flip}(a,i)) = 1$. Let $Y(a)$ be the set of all $i$'s such that some term $t$ is almost satisfied by $a$ with respect to $i$. The following lemma gives a bound on the size of $Y(a)$:

**Lemma 2 ([AP92])** *Let $F$ be an RkSj formula, and let $a \in \{0,1\}^n$ such that $F(a) = 1$. Then, $|Y(a)| \leq 2kj$.*[1]

**Corollary 3** *For all $a$ such that $F(a) = 0$, at most $2kj$ of the neighbors of $a$ (in the cube $\{0,1\}^n$) are satisfying assignments of $F$.*

**Proof:** The neighbors of $a$ in the cube are exactly $\mathit{flip}(a,1), \ldots, \mathit{flip}(a,n)$. Hence if $\mathit{flip}(a,i)$ is a satisfying assignment then $i \in Y(a)$. ■

Let $\mathcal{U}$ denote the uniform distribution, and let "$x \in \mathcal{U}$" denote that $x$ is selected uniformly at random. Also, let "true" be the "all 1's" concept.

**Lemma 4** *If $F$ is an RkSj DNF, and $F \not\equiv$ true, then* $\mathbf{Prob}_{x \in \mathcal{U}}[F(x) = 0] \geq 2^{-2kj}$

**Proof:** Think of the cube $\{0,1\}^n$ as a graph in which each vertex is represented by a $n$-bit string and there is an edge between any two vertices of Hamming distance 1. Consider the vertex-induced subgraph $G = (V, E)$, where $V$ is the set of points $a \in \{0,1\}^n$ such that $F(a) = 0$. Since $F \not\equiv$ true, we know that there is at least one node in $V$. We further know by Corollary 3 that the minimum degree of any node in $G$ is at least $n - 2kj$.

A simple argument shows that any subgraph of $\{0,1\}^n$ with minimum degree at least $d$ has at least $2^d$ nodes (e.g., [Sim83]). For completeness we give the proof: by induction on $n$. $n = 1$ is easy to verify. For larger $n$ if $d = 0$ it is easy again. Otherwise there is a coordinate $i$ in which if we cut the cube in coordinate $i$ into two sub-cubes of dimension $n-1$, then in each of them the minimum degree is at least $d - 1$. This is because each node has at most one neighbor in the other sub-cube. Also, by the choice of $i$ we have a non-empty subset of $V$ in each sub-cube. Using the induction we get at least $2 \cdot 2^{d-1} = 2^d$ nodes.

This implies that $|V| \geq 2^{n-2kj}$, which implies that $\mathbf{Prob}_{x \in \mathcal{U}}[F(x) = 0] \geq 2^{-2kj}$ ■

**Lemma 5** *Let $F$ be an RkSj formula. Then $F$ has at most $kjq$ terms of length (number of literals) at most $q$.*

**Proof:** Consider a graph with one vertex for each term in $F$ of length at most $q$ and an edge between two vertices if their corresponding terms are not simultaneously satisfiable. That is, if there exists $i$ such that one of the terms contains $x_i$ and the other contains $\overline{x_i}$. The maximum degree of a vertex in this graph is $q(k-1)$. (The reason is that every neighbor of a vertex must be a term that shares (at least one) variable with it and $F$ is Read-$k$.) Therefore, if the graph has $N$ vertices, it must have an independent set of size at least $N/(q(k-1)+1)$: such an independent set could be formed by repeatedly choosing a vertex to put in the set, and then eliminating from consideration all of the at most $q(k-1)$ neighbors of the vertex. On the other hand, the size of any independent set in this graph can be at most $j$ since $F$ is a Satisfy-$j$ DNF (any independent set of vertices corresponds to terms that can all be simultaneously satisfied). Therefore $N$, the number of terms of length at most $q$, is at most $j(q(k-1)+1) \leq jkq$. ■

**Definition 6** *A term $\tilde{t}$ is a $p$-variant of a term $t$ if it contains all the literals of $t$ and at most $p$ additional literals.*

**Lemma 7 ([AP92])** *Let $a$ be a satisfying assignment of an RkD formula $F$ and assume $a$ satisfies a term $t$ of $F$. Then $t$ is a $2k$-variant of $\mathit{term}(\mathit{sensitive}(a))$.*

The next lemma is a variant of Lemma 8 from [AP92]. The new version is just easier for use in our arguments. For completeness we give the proof here.

**Lemma 8** *If $a$ is a satisfying assignment for an RkSj formula $F$, then either some term satisfied by $a$ is a $4kj$-variant of $\mathit{term}(\mathit{sensitive}(a))$ or else there is a set of literals $U$, such that $|U| \geq 2kj$ and for all $\ell \in U$, the assignment $b = \mathit{flip}(a,\ell)$ is a positive example satisfying a strict subset of the terms of $F$ satisfied by $a$.*

**Proof:** Let $T$ be the set of terms satisfied by $a$. If some term in $T$ is a $4kj$ variant of $\mathit{term}(\mathit{sensitive}(a))$ then we are done. Otherwise, let $t$ be any term in $T$, and let $W = \{w_1, w_2, \ldots, w_{4kj}, \ldots\}$ be the set of at least $4kj$ literals that are in $T$ but are not in $\mathit{sensitive}(a)$. By Lemma 2 for at least $2kj$ literals in $W$ there does not exist a term in $F$ almost satisfied by it. Let this set be $U$. We get that for all $\ell \in U$ the assignment $\mathit{flip}(a,\ell)$ does not satisfy any term not satisfied by $a$. As such an assignment does not satisfy the term $t$, it satisfies a strict subset of $T$. ■

## 4 The Learning Algorithm

Our algorithm has the following high level structure (see Figure 1), which we describe here. The low level of the algorithm is described in the next two sub-sections.

Let $F$ be the target function and $F = t_1 \vee t_2 \vee \ldots \vee t_s$ its RkSj representation. The main tools in the algorithm are procedures *produce-terms* and *find-useful-example* that together, given a positive example $b$ of $F$ that satisfies, say, the term $t_1$, find a set of terms such that with high probability one of them is a prime implicant of the term $t_1$. The disjunction of all the terms found in this way is used as the hypothesis of the algorithm,

[1] Note that the lemma was stated originally only for satisfying assignments [AP92], however this assumption is not used in the proof and the statement of the lemma holds for every assignment $a \in \{0,1\}^n$.

with which we now ask an equivalence query (Step (2)). A negative counterexample allows us to throw out from the hypothesis terms that do not imply $F$. A positive counterexample satisfies a term $t_i \in F$ we have not yet found, and we use it to run again the procedures *find-useful-example* and *produce-terms*.

### 4.1 Producing terms

We now describe the way the procedure *produce-terms* works given a positive example $x$ of $F$. In our analysis, we will assume that example $x$ has the property that some term $t_1$ of $F$ satisfied by $x$ is a $4kj$-variant of $term(sensitive(x))$. This holds true for disjoint DNF by Lemma 7. For general RkSj DNF, we use procedure *find-useful-example* (described in Section 4.2) that, given any positive example, produces a set of examples such that with high probability at least one has this desired property. The procedure *produce-terms* is described in Figure 2.

Our goal is to find an implicant of $F$ that is implied by $t_1$. In fact, we will find a "small" collection of potential terms such that one of them will be such an implicant. We start by finding $sensitive(x)$, and we then fix those literals for the remainder of the procedure. After fixing (projecting onto) those literals, we still have an RkSj DNF, but $t_1$ has at most $4kj$ literals by our assumption. Let $t = \ell_1 \ell_2 \cdots \ell_r$ ($r \leq 4kj$) be some minimal (prime) implicant of $F$ implied by $t_1$. Note that this means that for each $i \in \{1, 2, \ldots, r\}$ the term $t^{(i)} = \ell_1 \cdots \ell_{i-1} \overline{\ell_i} \ell_{i+1} \cdots \ell_r$ is *not* an implicant of $F$, because then we could remove $\ell_i$ from $t$ and obtain an implicant contradicting the primality of $t$.

The idea is that we create a set of literals $L$ that with high probability has the following two properties: (1) $L$ has all the literals of $t$, and (2) $L$ has size poly$(kj)$. This is done in Step (3). If we have such a set then we are done: given the set $L$, we can look at all $\sum_{i=0}^{4kj} \binom{|L|}{i} = |L|^{O(kj)}$ small subsets (Step (4)), and one of them will be $t$.

Recall that $s$ is the number of terms in the RkSj representation of $F$, and let $\delta$ be a given confidence parameter. We will show that the algorithm finds a function logically equivalent to $F$ with probability at least $1 - \delta$.

**Claim 9** *If $x$ is a positive example with the property described above, then the set $L$ produced in Step (3) of procedure produce-terms contains all the literals in $t$ with probability at least $1 - \delta/3s$.*

**Proof:** Consider a literal $\ell_i$ in $t$. In a random $y$, there is a $1/2^r$ chance that literals $\ell_1, \ldots, \ell_r$ are set so that $t^{(i)}$ is satisfied. Also,

$$\mathbf{Prob}_{y \in \mathcal{U}}[y \text{ is negative} \mid y \text{ satisfies } t^{(i)}] \geq 1/2^{2kj}.$$

The reason is that if we project variables in $t$ to satisfy $t^{(i)}$, then we are left with an RkSj DNF which is not identically `true`, since $t^{(i)}$ is not an implicant of $F$. So, we can apply Lemma 4. For such an example $y$, we know that $flip(y, \ell_i)$ is a positive example (as it satisfies $t$) and hence $\ell_i$ will be inserted into $L$. So, the probability that a single $y$ "discovers" $\ell_i$ is at least $1/2^{r+2kj} \geq 1/2^{6kj}$. A standard calculation shows that if we repeat the loop (Step (3)) at least $m_1 = 2^{6kj} \ln(12kjs/\delta)$ times then we find *all* $r \leq 4kj$ literals of $t$, with probability at least $1 - \delta/3s$. (Note that the number of repetitions, $m_1$, is polynomial as long as $kj = O(\log n)$.) ■

**Claim 10** *Let $q = c \log^2 n$ for $c = O(\log(s/\delta))$, and assume that $kj = O(\log n)$. With probability at least $1 - \delta/3s$ we get $|L| \leq kjq^2$.*

**Proof:** Any literal found in Step (3) of *produce-terms* must be in *some* term. So, consider terms of size at most $q$. By lemma 5 there are at most $kjq$ such terms. The number of literals that those terms can contribute to $L$ is at most $kjq^2$. Now consider a term of size greater than $q$. With high probability, it will not contribute *any* literals to $L$. The reason is that in order for such a term to "matter" (i.e. for some $flip(y, \ell)$ to satisfy that term) it must be that $y$ satisfies all but one literal in that term. The chance that $y$ does so is at most $n/2^q$ which, by the choice of $q$, is smaller than any polynomial fraction. Since the algorithm repeats in Step (3) a polynomial number of times (using here that $kj = O(\log n)$), throughout the entire algorithm, with high probability, *no* example $flip(y, \ell)$ satisfies *any* of those terms. By choice of $c$ in the specified range we can make this high probability be at least $1 - \delta/3s$. ■

Notice that by Claims 9 and 10, with high probability at least one of the terms output in Step (4) of the procedure *produce-terms* is the desired term $t$, and in addition at most $(kjq^2)^{4kj}$ terms are output in Step (4) total. This number of terms is polynomial in $n$ as long as $kj = O(\log n / \log\log n)$.

### 4.2 Finding a "useful" positive example.

The procedure *produce-terms* in the previous section requires a positive example $x$ that satisfies some term $t_1$ of $F$ that is (1) not in our hypothesis and (2) is a $4kj$-variant of $sensitive(x)$. We call such an example $x$ a "useful" example. In this section we show how to produce, given a positive counterexample $x$ to our hypothesis, a polynomial-sized set of examples such that with high probability at least one is "useful" in this sense. The procedure is described in Figure 3. The idea behind procedure *find-useful-example* is as follows. Suppose that our given positive example $x$ satisfies some number of terms of $F$ (none of them is satisfied by our hypothesis). Lemma 8 states that either:

(A) All but $4kj$ of the literals in one of those terms are sensitive (i.e., $x$ is already "useful"), or else,

(B) There exist at least $2kj$ literals $u_1, \ldots, u_{2kj}$ such that flipping $u_i$ will cause the example to remain positive but satisfy a strict subset of those terms and no others.

**Algorithm** *Learn-RkSj*:

1. $H \leftarrow \phi$
2. $b \leftarrow EQ(\vee_{h \in H} h)$.
3. if $b$ ='true' then stop and output $H$.
4. Else, if $b$ is a negative counterexample then for every term $t \in H$ such that $t(b) = 1$ delete $t$ from $H$.
5. Else (if $b$ is a positive counterexample) then
   (a) $S \leftarrow$ *find-useful-example*$(b)$. /* This procedure returns a set of examples. */
   (b) For each $a \in S$, $H \leftarrow H \cup$ *produce-terms*$(a)$.
6. GOTO 2

Figure 1: The Algorithm *Learn-RkSj*

**Procedure** *produce-terms*$(x)$:

1. Find $sensitive(x)$ (using $n$ membership queries).
2. $L \leftarrow \emptyset$.
3. Repeat the following $m_1 = 2^{6kj} \ln(12kjs/\delta)$ times:
   Pick a random $y$ that agrees with $sensitive(x)$ (that is, for each literal in $sensitive(x)$ we take $y_i = x_i$ and for any other $i$, $y_i$ is a random bit). If $y$ is negative (a membership query) then find all literals $\ell$ satisfied by $y$ (and not in $sensitive(x)$) such that $flip(y, \ell)$ is a positive example. Put all those literals into $L$.
4. If $|L| \leq kjq^2$, then for each subset $L'$ of $L$ of size at most $4kj$, add to the output the term $term(sensitive(x) \cup L')$.

Figure 2: The Procedure *produce-terms*

**Procedure** *find-useful-example*$(x)$:
Do the following $m_2 = 2^{j+1} \log(3s/\delta)$ times and produce the union of the outputs.

1. Let $x^{(0)} = x$.
2. For $i = 1, 2, \ldots, m_3$, for $m_3 = \frac{n}{2k} \ln(2j2^j)$, let $x^{(i)}$ be a random positive neighbor of $x^{(i-1)}$.
3. Output $\{x^{(0)}, x^{(1)}, \ldots, x^{(m_3)}\}$.

Figure 3: The Procedure *find-useful-example*

We also know that since $|Y(x)| \leq 2kj$ by Lemma 2, there are at most $2kj$ literals that when flipped cause the example to satisfy some additional term.

What this means is that either example $x$ is already useful, or else out of the $n$ literals, at least $2kj$ are "good" in that flipping them makes our example satisfy a strict subset of the original set of terms, at most $2kj$ are "bad" in that flipping them makes the example satisfy some new term, and the rest are "neutral" in that either flipping them makes the example negative (which we will notice) or else flipping them does not affect the set of terms satisfied. So, as described in Figure 3 (Step (2)), let $x^{(i)}$ to be a random positive neighbor of $x^{(i-1)}$ in the boolean hypercube.

**Claim 11** *With probability $\frac{1}{2^{j+1}}$ at least one of the examples $x, x^{(1)}, \ldots, x^{(m_3)}$ is useful.*

**Proof:** For the moment, consider the infinite sequence $x^{(1)}, x^{(2)}, \ldots$. If the example $x$ was not already useful, let's say that we are "lucky" if we flip one of the "good" literals before we flip any of the "bad" literals in this experiment. Notice that flipping a "neutral" bit may change the sets of good and bad literals, but our bounds on the sizes of those sets remain. So, the probability we are lucky is at least 1/2. If this occurs, we then either have a useful positive example, or else our bounds on the sizes of the good and bad sets remain but the example satisfies at least one fewer term. Thus, with probability at least $(1/2^j)$, we continue to flip good literals before we flip any bad bit until we reach a useful example in this experiment.

Now for the finite sequence $x^{(1)}, \ldots, x^{(m_3)}$, we have to subtract the probability of being lucky $j$ times, but not within the first $m_3$ trials. This event is included in the event "there were less than $j$ flips of non-neutral literals within $m$ trials". Let the latter event be $E$.

Consider $m_3$ as $j$ blocks of $\frac{n}{2kj}\ln(2j2^j)$ trials each. The probability that we do not flip any non-neutral bit (or reach a useful example) in one block is at most $\frac{1}{2j\,2^j}$ (since the probability of getting a non-neutral bit is at least $\frac{2kj}{n}$ at each trial). The probability that we fail in any of the $j$ blocks is therefore at most $\frac{1}{2^{j+1}}$. This is also a bound for the probability of the event $E$. Thus our total success probability is at least $\frac{1}{2^j} - \frac{1}{2^{j+1}}$, which proves the claim. ■

Notice that for $j = O(\log n)$ the success probability in Claim 11 is $1/poly(n)$.

**Claim 12** *With probability at least $1 - \delta/3s$, some example in the set of examples produced by the procedure find-useful-example is "useful."*

**Proof:** Procedure *find-useful-example* repeats the experiment described in Claim 11 for $m_2 = 2^{j+1}\log(3s/\delta)$ times. Thus, with probability at least $1 - \delta/3s$, the experiment is successful at least once. ■

### 4.3 Final analysis

**Claim 13** *The algorithm Learn-RkSj uses $O(snm_1m_2m_3)$ membership queries and $O(sm_2m_3(kjq^2)^{4kj})$ equivalence queries.*

**Proof:** In each call to *produce-terms* we make $n$ membership queries to find $sensitive(x)$, one membership query for each $y$, and $n$ additional queries for each $y$ that is negative. All together the procedure *produce-terms* makes $O(nm_1)$ membership queries to create the set of literals $L$. It then produces (in Step (4)) $O((kjq^2)^{4kj})$ terms (or none, in the unlikely event that $|L| > kjq^2$). The number of calls to the procedure *produce-terms* is bounded by the number of positive counterexamples that we get in the algorithm (which is bounded by $s$) multiplied by the number of examples produced by the procedure *find-useful-example* on each such counterexample (this is bounded by $m_2m_3$). Therefore, in total we make $O(snm_1m_2m_3)$ membership queries and produce $O(sm_2m_3(kjq^2)^{4kj})$ terms. This immediately gives a bound on the number of negative counterexamples. Hence, the number of equivalence queries is $O(sm_2m_3(kjq^2)^{4kj})$. Note that just by the Read-$k$ property of $F$ we have $s \leq kn$. A tighter bound of $s = O(\sqrt{k^2jn})$ is given by [Mat93]. ■

**Theorem 14** *For $\delta > 0$ and $kj = O(\frac{\log n}{\log\log n})$ the algorithm runs in time $poly(n, \ln\frac{1}{\delta})$ and finds a function logically equivalent to $F$ with probability at least $1 - \delta$.*

**Proof:** By Claim 12, each time that procedure *find-useful-example* is invoked (on a positive counterexample), we get a useful example with probability at least $1 - \delta/3s$. By Claim 9 and Claim 10, if we have a useful example then we find a term $t$ that satisfies the original counterexample with probability at least $1 - 2\delta/3s$. All together we find a good term $t$ with probability at least $1 - \delta/s$. Therefore, if we get $s$ positive counterexamples, with probability at least $1 - \delta$, we find prime implicants for all the terms in $F$, and therefore the hypothesis is logically equivalent to $F$. The polynomial bound then follows from Claim 13, noting that $m_1, m_2$ and $m_3$ are all polynomial for $kj = O(\log n)$ and that $(kjq^2)^{O(kj)}$ is polynomial for $kj = O(\log n/\log\log n)$ (as $q$ is $polylog(n)$). ■

As a last remark we note that it may be that the method we use here can be adapted for $kj = O(\log n)$. The only part of the algorithm that does not allow for $kj = O(\log n)$ is the procedure *produce-terms* and in particular Step (4) that produces $O((\log n)^{O(kj)})$ terms. Finding a better sampling method or a better way to produce the terms out of the set $L$ might solve this problem.

## 5 Read-2 Disjoint DNF does not have small CNF

We present a family of functions, $\{F_n\}$ that have short ($poly(n)$) R2D DNF formulae but require CNF formulae

of size $2^{\Omega(\sqrt{n})}$. These functions were defined in [AP92], where it was shown that the class R2D is not included in Read-$k$ decision trees. The result was later strengthened in [Aiz93] to show that the family requires decision trees of size $2^{n-2}$. Thus, a polynomial time algorithm for learning decision trees does not subsume the work presented here. Recently, [Bsh93] gives an algorithm for learning functions which have both small DNFs **and** small CNFs. We show that the algorithm of [Bsh93] does not apply here, as the size of the CNFs for even read-2 disjoint DNFs must be super-polynomial. It remains open, however, whether techniques similar to those used in [Bsh93], can be applied to yield a comparable, or stronger result.

**Theorem 15** *$R2D \not\subseteq CNF - size(2^{\sqrt{2n}} - 2(\sqrt{2n}+1))$.*

**Proof:** Let $n = \binom{m+1}{2}$. We define a function $F_n$ with $m+1$ terms each of length $m$ on $n$ variables $x_{i,j}$ where $1 \leq i < j \leq m+1$. The R2D representation of the function is $t_1 \vee t_2 \vee \ldots \vee t_{m+1}$, where the term $t_q$ includes all the literals $x_{q,j}$ for $j > q$, and all the literals $\overline{x_{j,q}}$ for $j < q$. The idea is that the variable $x_{i,j}$ appears only in the $i$-th term and the $j$-th term, and is "responsible" for the disjointness of these two terms (since the variable appears negated in one term, and un-negated in the other). For example, for $m = 3$ the function is: $x_{12}x_{13}x_{14} \vee \overline{x_{12}}x_{23}x_{24} \vee \overline{x_{13}}\,\overline{x_{23}}x_{34} \vee \overline{x_{14}}\,\overline{x_{24}}\,\overline{x_{34}}$. The main step in the proof is the following claim:

**Claim 16** *Every clause in a CNF representation of $F_n$ must have $m+1$ literals.*

Given this claim, the following counting argument shows that the CNF must have a large number of clauses: The number of assignments satisfied by $F_n$ is exactly $(m+1)2^{n-m}$, since each of the $m+1$ terms satisfies $2^{n-m}$ assignments (it determines the value of $m$ out of the $n$ variables) and the representation is disjoint. This implies that the number of assignments unsatisfied by $F_n$ is $2^n(1-(m+1)/2^m)$. Since a clause of length $\geq m+1$ falsifies at most $2^{n-m-1}$ assignments, we need at least $2^{m+1}(1-(m+1)/2^m) = 2^{m+1} - 2(m+1)$ clauses to falsify all the unsatisfying assignments of $F_n$. As $m^2 \leq 2n \leq (m+1)^2$ the theorem follows.

**Proof of the Claim:**
Assume by way of contradiction, that there is a clause of length $\alpha < m+1$ in a CNF representation for $F_n$. Let $C = (\ell_1 \vee \ell_2 \vee \cdots \vee \ell_\alpha)$ be that clause. We show that there exists an assignment $a$ such that $C(a) = 0$ (and hence the value of the CNF formula on $a$ is 0) and $F_n(a) = 1$, contradicting the assumption that $C$ is a clause in a CNF representation of $F_n$.

To construct $a$ notice that since every *literal* appears exactly once in the R2D representation, fixing the values of $\alpha \leq m$ literals can falsify at most $m$ terms in the DNF representation, so there is still a term $t_C$ that can be satisfied. Therefore, we can define $a$ to be the assignment in which all the literals in $C$ are set to zero, all the literals in $t_C$ are set to one, and all other literals are set arbitrarily. We thus have that $C(a) = 0$ but $t_C(a) = 1$ (and hence $F_n(a) = 1$), a contradiction. ■

## References


[AFP92] D. Angluin, M. Frazier, and L. Pitt. Learning conjunctions of Horn clauses. *Machine Learning*, 9:147–164, 1992.

[Aiz93] H. Aizenstein. *On the Learnability of Disjunctive Normal Form Formulas and Decision Trees.* PhD thesis, Dept of Computer Science, University of Illinois, 1993. Technical report UIUCDCS-R-93-1813, June, 1993, Urbana, Illinois.

[AK91] D. Angluin and M. Kharitonov. When won't membership queries help? In *Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing*, pages 444–454, New Orleans, Louisiana, May 1991.

[Ang88] D. Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, April 1988.

[AP91] H. Aizenstein and L. Pitt. Exact learning of read-twice DNF formulas. In *Proceedings of the IEEE Symp. on Foundation of Computer Science*, number 32, pages 170–179, San Juan, 1991.

[AP92] H. Aizenstein and L. Pitt. Exact learning of read-$k$ disjoint dnf and not-so-disjoint dnf. In *Proceedings of COLT '92*, pages 71–76, Pittsburgh, Pennsylvania, 1992. Morgan Kaufmann.

[AP93] H. Aizenstein and L. Pitt. DNF cannot be learned by greedily collecting prime implicants. Unpublished, 1993.

[BFJ+94] A. Blum, M. Furst, J. Jackson, M. Kearns, Y. Mansour, and S. Rudich. Weakly learning DNF and characterizing statistical query learning using fourier analysis. In *Proceedings of Twenty-sixth ACM Symposium on Theory of Computing*, 1994. To appear.

[BR92] A. Blum and S. Rudich. Fast learning of $k$-term DNF formulas with queries. In *Proceedings of Twenty-Fourth ACM Symposium on Theory of Computing*, pages 382–389. ACM, 1992.

[Bsh93] N. H. Bshouty. Exact learning via the monotone theory. In *Proceedings of the IEEE Symp. on Foundation of Computer Science*, pages 302–311, Palo Alto, CA., 1993.

[Han91] T. Hancock. Learning $2\mu$ DNF formulas and $k\mu$ decision trees. In *Proceedings of the Fourth Annual Workshop on Computational Learning Theory*, pages 199–209, Santa Cruz, California, August 1991. Morgan Kaufmann.

[Kha94] R. Khardon. On using the Fourier transform to learn disjoint DNF. *Information Processing Letters*, 49(5):219–222, March 1994.

[KM93] E. Kushilevitz and Y. Mansour. Learning decision trees using the fourier spectrum. *Siam Journal of Computing*, 22(6):1331–1348, 1993. Earlier version appeared in Proc. 23rd Ann. IEEE Symp. on Foundations of Computer Science, 1991.

[KR93] E. Kushilevitz and D. Roth. On learning visual concepts and dnf formulae. In *Proceedings of the ACM Workshop on Computational Learning Theory '93*, pages 317–326. Morgan Kaufmann, 1993.

[Mat93] S. Matar. Learning with minimal number of queries. Master's thesis, University of Albert, Canada, 1993.

[PR93] K. Pillapakkamnatt and V. Raghavan. Read twice dnf formulas are properly learnable. Technical Report TR-CS-93-59, Vanderbilt University, Computer Science Department, 1993. To appear, Proceedings of the 1st European Conference on Computational Learning Theory (EuroColt 93).

[Sim83] H. U. Simon. A tight $\omega(\log\log n)$-bound on the time for parallel ram's to compute nondegenerate boolean functions. In *ICALP*, number 158 in Lecture Notes in Computer Science, pages 439–444. Springer-Verlag, 1983.

[Val84] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, November 1984.