Exact Learning of subclasses of CDNF formulas with membership queries*

Carlos Domingo

Dept. of Computer Science,
Tokyo Institute of Technology,
Ookayama, Meguro-ku, Tokyo 152,
Japan
Email: carlos@cs.titech.ac.jp

Abstract. We consider the exact learnability of subclasses of Boolean formulas from membership queries alone. We show how to combine known learning algorithms that use membership and equivalence queries to obtain new learning results only with memberships. In particular we show the exact learnability of read-k monotone formulas, Sat-k $\mathcal{O}(\log n)$-CDNF, and $\mathcal{O}(\sqrt{\log n})$-size CDNF from membership queries only.

1 Introduction

Learning DNF formulas has been one of the most attractive and tantalizing problems since the seminal paper of Valiant [**Val84**]. Although many results in the literature gave evidence that the problem is hard even if we are allow to use membership queries [**AK91**, **AHP92**], it has been recently proved by Jackson [**Jac94**] that using membership queries, DNF are PAC learnable in polynomial time under the uniform distribution. Here we concentrate in a more restricted framework. While Jackson's algorithm is a PAC learning algorithm, we wish to have exact identification of the target formula. In this more restricted setting known as exact learning [**Ang88**] few positive results have been discovered using only membership queries queries. Among others, the following very restricted subclasses of monotone DNF formulas are properly exact learnable by membership queries: monotone read-once formulas [**AHK93**], k-term monotone DNF [**BGHM93**], and 2-monotonic positive functions [**MI94b**]. In the non monotone case, decision trees of $\mathcal{O}(\log n)$-depth [**KM93**] and Sat-k $\mathcal{O}(\log n)$-DNF [**BF+94**] are shown to be exact learnable using a threshold of parity functions as a hypothesis.

In almost all the cases, we show that stronger results hold if we measure the efficiency of the algorithm with respect to the DNF size as well as the CNF size. Our approach is the following. We use well known learning algorithms [**Ang88**, **Bsh95**] that learn certain classes of formulas with membership and equivalence queries. Those algorithms have some nice properties and we

* Research supported by the Esprit EC program under project 7141 (ALCOM-II) and Working Group 8556 (NeuroColt), and by the Spanish DGICYT (project PB92-0709).

show how to exploit them to learn some classes of formulas with only membership queries. Those algorithms are monotone with respect to the target, i.e. they always produce a hypothesis that implies the target. Moreover, they have a dual algorithm that behaves in a symmetric way. Thus, combining them we are able to obtain new positive learning results. The price we have to pay for learning only with memberships is that now, the query complexity depends on the DNF as well as the CNF size of the target, not only one of them as it would be desired.

Nevertheless, our results are optimal in the number of membership needed. It has been shown in [**BGHM93**] that any exact learning algorithm that uses only membership queries must consider simultaneously the DNF and the CNF size when measuring the number of membership queries for monotone formulas and the dual and monotone dual dimension (which in our case are polynomially related to the DNF and CNF size but not in general, see Section 4) for non monotone ones.

The classes we show to be exact learnable only with membership queries are read-k monotone formulas, Sat-k $\mathcal{O}(\log n)$-DNF formulas and $\mathcal{O}(\sqrt{\log n})$-size CDNF. For proving our results, we show how to test equivalence between some classes of Boolean formulas. These algorithms may be of independent interest outside the learning community.

The rest of the paper is organized as follows. Section 2 gives definition and terminology used throughout the paper. Section 3 shows how to learn read-k monotone formulas. In Section 4 the learnability of subclasses of general DNF formulas is shown. Finally, Section 5 discuss the results and presents some future work and open problems.

2 Preliminaries

We are interested in the learnability of Boolean functions. First, we give some definitions and notation. We denote by e_i the assignment that is 0 everywhere except in the i-th bit. A literal is either a variable x_i or its negation $\bar{x_i}$. We will use the order $\leq$ over $\{0,1\}^n$ where for two assignments a and b, $a \leq b$ if for every $1 \leq i \leq n$, $a_i \leq b_i$. A Boolean function is said to be *monotone* if for every pair of assignments $a, b \in \{0,1\}^n$, $a \leq b$ implies $f(a) \leq f(b)$. A Boolean function is called *unate* if there exists an assignment a such that $f(x+a)$ is monotone, where "+" is the group addition in $GF(2)^n$.

Now we define several classes of formulas that we will consider. A *DNF* expression is a disjunction of terms, where each term is a conjunction of literals and a *CNF* expression is a conjunction of clauses, where each clause is a disjunction of literals. The size of a DNF formula, denoted by $d(f)$, is the number of terms and the size of a CNF formula, denoted by $c(f)$ is the number of clauses. A k-term DNF (k-clause CNF) is the class of functions represented as a DNF (CNF) with at most k terms (clauses). We denote by k-size CDNF the class of formulas included in k-term DNF $\cap$ k-clause CNF. On the other hand, k-DNF (k-CNF) is the class of formulas that have at most k literals in each term (clause). The class Sat-k DNF (CNF) is the class of formulas where each assignment satisfies at most k terms (clauses) simultaneously. Furthermore, a Sat-k CDNF is a formulas that is contained in the class Sat-k DNF $\cap$ Sat-k CNF. Finally, the class of k-depth decision trees is the class of Boolean functions that can be represented as a decision tree of depth at most k. The class of k-depth decision trees is strictly included in the class Sat-1 k-CDNF, also called Disjoint k-CDNF.

We now recall the definitions of the Fourier transform needed in Section 4. Every Boolean function can be uniquely expressed in terms of its Fourier spectrum. That is, $f(x) = \sum_{S \subseteq \{1,\ldots,n\}} \hat{f}(S)\chi_S(x)$ where $\chi_S(x) = \prod_{i \in S}(-1)^{x_i}$ and the Fourier coefficients $\hat{f}(S)$ calculate the correlation of f and χ_S with respect to the uniform distribution. Furthermore, because the vector space of real-valued functions over $\{0,1\}^n$ has dimension 2^n, the set of 2^n parity functions forms an orthonormal basis for this vector space and every function f can be uniquely expressed as a linear combination of parity functions. We call the *support* of a function to $\{S|\hat{f}(S) \neq 0\}$, i.e. the set of subsets of variables such that the corresponding Fourier coefficient is non zero.

As we mention in the introduction, we are interested in learning algorithms that are "monotone" with respect to the target. The following definition formalizes this idea.

Definition 1. A learning algorithm L is said to be *monotone* if learning a target formula f, always produce hypothesis h such that $h \Rightarrow f$. Symmetrically, a learning algorithm L^δ is *antimonotone* if always produce hypothesis h^δ such that $f \Rightarrow h^\delta$.

```
algorithm $L \bowtie L^\delta$
  run $L$ until it produces $h$
  run $L^\delta$ until it produces $h^\delta$
  while $EQU(h, h^\delta, c) \neq$ "$YES$" do
  /* $EQU$ checks whether $h \equiv h^\delta$, and if not returns $c$ such that $h(c) \neq h^\delta(c)$ */
    if $h(c) \neq f(c)$ then
      give $c$ to $L$ and update $h$.
    else
      give $c$ to $L^\delta$ and update $h^\delta$.
    end
  end
```

Fig. 1. Combination of a monotone and an antimonotone learning algorithm.

Therefore, we can combine both algorithms as it is shown in Figure 1 and obtain the following useful tool for designing new learning algorithms.

Lemma 2. *Let L and L^δ be respectively a monotone and antimonotone exact learning algorithms with membership and equivalence queries for a class of concepts $\mathcal{C}$. Let $\mathcal{H}$ and $\mathcal{H}^\delta$ be the hypothesis class used by L and L^δ respectively and let EQU be and algorithm for decide the equivalence between any pair of formulas $h \in \mathcal{H}$ and $h^\delta \in \mathcal{H}^\delta$ such that $h \Rightarrow h^\delta$. Then, the concept class $\mathcal{C}$ is exactly learnable only with membership queries.*

We observe that even if algorithm EQU just decides whether h is equivalent to h^δ without finding the counterexample c needed in our learning algorithm, n extra calls of EQU suffices to find a counterexample. Notice also that the monotonicity restriction of the algorithms can be relaxed as follows. We only need that both algorithms never issue a hypothesis equal to any hypothesis issued by the other except when they find the target. However, we have not found any pair of learning algorithms that satisfy this condition and are not "monotone" in the previous sense.

This way to combine learning algorithms was used in [**BC+95**] to obtain learning algorithms with membership queries and an NP-oracle. In this paper, we concentrate in some classes that can be effectively learned with membership queries without the need of an NP-oracle.

3 Learning monotone formulas

The first algorithm we are going to use is Angluin's algorithm (denoted by A) for learning monotone DNF formulas with membership and equivalence queries [**Ang88**]. Algorithm A is a monotone learning algorithm and the hypothesis produced are always monotone DNF formulas. Moreover, it is easy to see that we can invert algorithm A by greedily collecting all maximal false vectors from negative counterexamples instead of greedily collection all minimal true assignments from positive counterexamples. Thus, we obtain the dual algorithm A^δ that learns monotone CNF formulas with membership and equivalence queries and hypothesis that are monotone CNF formulas. Moreover, this algorithm is an antimonotone algorithm. Thus, algorithms A and A^δ meet the requirements of Lemma 2. In the following Subsection, we show an interesting class that is learnable by the combination of A and A^δ. This way of combining Angluin's algorithm was used in [**BI94**] when comparing learning with dualization.

3.1 Learning Read-k monotone formulas

A read-k DNF formula is a formula where each variable appears at most k times. The *sensitive set* of an assignment x is defined by $S(x) = \{y | f(x) \neq f(y), d(x,y) = 1\}$ where $d(x,y)$ is the Hamming distance of x and y. If $S(x) \neq \emptyset$ and $f(x) = 1$ then we say that x is a *positive sensitive assignment* of f. Let us first give a couple of useful properties of read-k formulas.

Proposition 3. *If x is a positive sensitive assignment of read-k monotone DNF formula f then x simultaneously satisfies at most k terms of f.*

Proof. Let x be a positive sensitive assignment that satisfies $k+1$ terms of a read-k DNF formula f and let $y \in S(x)$ such that y just differs from x in the i-th bit. Since flipping the i-th bit of x from 1 to 0 the formula is not satisfied, x_i must appear in the $k+1$ terms of f that x satisfied. Thus, f is not a read-k formula. □

Proposition 4. *Let f be a read-k monotone DNF formula and let g be a monotone CNF formula. Assume that f is not equivalent to g and there exists an assignment x such that $f(x) = 0$ and $g(x) = 1$. Then, there exists another assignment y such that $x \leq y$, $f(y) = 0$, $g(y) = 1$ and y belongs to the sensitive set of a positive sensitive assignment of f.*

Proof. Let x be an assignment such that $f(x) = 0$ and $g(x) = 1$. We flip 0 bits of x while keeping $f(x) = 0$ until we obtain y such that $x \leq y$ and flipping one bit more of y we satisfy f. By monotonicity, g is still satisfied by y and y belongs to the sensitive set of a positive assignment for f. □

Using these two properties we can show how to check equivalence for a monotone read-k DNF formula.

Lemma 5. *Let f be a read-k monotone DNF formula and g a monotone CNF formula. We can test whether f and g are equivalent in polynomial time with respect to $d(f)$ and $c(g)$.*

Proof. Since f is in DNF and g in CNF we can efficiently check whether there exists an assignment x such that $f(x) = 1$ and $g(x) = 0$. If we find such an assignment we are done. Otherwise, if f and g are still not equivalent, there should be an assignment x such that $f(x) = 0$ and $g(x) = 1$. By Proposition 4 there exists another assignment y that belongs to the sensitive set of a positive sensitive assignment and is still a valid witness of non equivalence between f and g. Moreover, Proposition 3 tell us that all the positive sensitive assignment satisfy at most k terms. Therefore, we check the sensitive sets of all the possible positive sensitive assignments of f. The number of sensitive assignments of f is smaller than n^{k+1} which is polynomial for constant k. Thus, either we find an assignment y which proofs that f is not equivalent to g or, by Proposition 4 it does not exist and f is equivalent to g. □

Therefore, Lemma 2 together with algorithms A and A^{δ} and the above Lemma implies the following Theorem.

Theorem 6. *The class of read-k monotone DNF formulas is exactly learnable using membership queries in polynomial time in $max\{d(f), c(f)\}$.*

By results of [**BC+95**] this upper bound on the number of membership queries cannot be further improve. Moreover, since learning a read-k DNF for $k \geq 3$ is equivalent to learn an arbitrary DNF [**AK91**], the monotonicity restriction is also optimal. Observe that if we can learn non-monotone read-thrice formulas with membership queries we will be able to learn DNF formulas with membership queries and that is impossible since they contain the class of singletons (monomials of n variables). This class is not exactly learnable using less than 2^n membership queries [**Ang88**].

4 Learning arbitrary formulas

In [**Bsh95**], Bshouty showed that CDNF formulas are exactly learnable in polynomial time with membership and equivalence queries. He uses an algorithm, that we will call B which is based on Angluin's algorithm for monotone DNF formulas. Thus, it shares the same property, it is a monotone learning algorithm. The hypothesis are depth-3 formulas consisting of a CNF of polynomially many DNF formulas as clauses. The algorithm runs in polynomial time in the DNF size of the target and its monotone dimension (see [**Bsh95**] for a detailed definition). Moreover, his algorithm can be inverted obtaining a dual monotone learning algorithm B^δ for the same class of formulas [**BC+95**]. In this case the hypothesis are also depth-3 formulas but consisting of a DNF of polynomially many CNF formulas as terms. This dual version runs in polynomial time in the CNF size and its dual monotone dimension. In the following Subsections, we show some classes of formulas that are learnable by the combination of B and B^δ.

4.1 Learning Sat-k $\mathcal{O}(\log n)$-DNF formulas

In [**KM93**], Kushilevitz and Mansour gave an algorithm for exactly finding the support of the class of $\mathcal{O}(\log n)$-depth decision tree using membership queries. Moreover, in [**BF+94**] it was proved that the same algorithm also obtains the support of k-disjoint $\mathcal{O}(\log n)$-DNF formulas, a class that strictly contain decision trees of $\mathcal{O}(\log n)$-depth. These results are stated in the following theorem.

Theorem 7. *[**KM93**, **BF+94**] There is a deterministic algorithm, that for any Boolean function f that can be represented as a Sat-k $\mathcal{O}(\log n)$-DNF , outputs the support of f in polynomial time in n.*

We use previous algorithm to obtain the following result.

Lemma 8. *Let f and g be Boolean formulas that have a representation as a Sat-k $\mathcal{O}(\log n)$-DNF. Then, we can test in polynomial time in n whether f and g are equivalent.*

Proof. We just have to use the algorithm of Theorem 7 to obtain the supports of f and g, which have a polynomial number of sets. We check whether the supports of f and g are equal and answer 'yes' in case they are and 'no' otherwise. Since no two different sets of Fourier coefficients can represent the same Boolean function, the correctness follows. $\square$

Notice that we do not care about how formulas f and g are given in the previous lemma, as far as the could be represented as required. Thus, algorithm of Theorem 7 can be applied, since it just tests membership of the input formulas. It does not use any property of the way in which the input formulas are actually represented. Therefore, we can obtain the following learning result.

Theorem 9. *The class of Sat-k $\mathcal{O}(\log n)$-DNF formula are exactly learnable in polynomial time with respect to its DNF and CNF size, from membership queries alone. The hypothesis is a depth-3 formula.*

Proof. Run algorithm of Figure 1 using Bshouty's algorithms B and B^δ for CDNF formulas as algorithms L and L^δ and using the algorithm described in Lemma 8 for testing equivalence. Since that algorithm does not provide any counterexample, we need to run it $n+1$ times for solving every equivalence query. Moreover, this classes have monotone and dual monotone dimension polynomial (see [**Bsh95**] and [**BGHM93**]) and therefore the running time of B and B^δ is also polynomial. $\square$

The class of Sat-k $\mathcal{O}(\log n)$-DNF contains decision trees of logarithmic depth, so we also exactly learn the same class. Notice that we cannot extend this result to any class of decision trees that can represent singletons (for instance, polynomial size decision trees) since that class is known to be non exactly learnable with polynomial number of membership queries. As we mentioned before, our result was already proved in [**BF+94**]. However, by using the combination of Bshouty's algorithms, we are able to improved the way the hypothesis is represented. Our learnability result is in terms of a depth-3 formula while the result in [**BF+94**] was in terms of a threshold of parity functions (a Fourier based representation). It is open whether this class of formulas is proper learnable in polynomial time from membership queries only.

4.2 Learning Bounded size CDNF formulas

In this subsection we show how to use again the combination of Bshouty's algorithms to learn a class of formulas that have bounded number of terms and clauses in its DNF and CNF representation respectively. First, let see that we can test equivalence between some formulas.

Lemma 10. *Let $f = f_1 \wedge \ldots \wedge f_r$ and $g = g_1 \vee \ldots \vee g_s$ be depth-3 formulas, where f_i are DNF formulas and g_j are CNF formulas such that $|f| + |g| = \mathcal{O}(\log n)$ and $f \Rightarrow g$. Then, we can test whether f is equivalent to g in polynomial time in n.*

Proof. Since f implies g, there always exists at least one variable x_k that either appears positive or negative in both formulas simultaneously. To see this claim, observe that when $f(x) = 1$, we just have to set the variables in some terms $t_{i_1} \in f_1, t_{i_2} \in f_2, \ldots, t_{i_r} \in f_r$ such that all of them are satisfied, i.e. $(t_{i_1}(y) \wedge$

$\ldots \wedge t_{i_r}(y)) = 1$, where y is a partial assignment contained in x but defined only with the variables that appear in those terms. The partial assignment y should already satisfy one $g_j \in g$, i.e. $g_j(y) = 1$. Thus, it should be at least one variable x_k that appears either positive or negative simultaneously in a one of the clauses satisfied by y and in g_j. Then, we build a decision tree with x_k in the root where both its children have either a term less in f or a clause less in g and we solve the problem recursively. If all the leaves of this tree have equivalent pair of formulas, the input formulas are equivalent. However, if we find a witness of non equivalence in a pair of formulas in any leaf, we can carry back that witness and show the non equivalence of the input formulas. After at most $\mathcal{O}(\log n)$ iterations either f or g becomes constant. Suppose that g becomes constant in one of the leaves. Then, if there is an assignment y to the variables of function f in that leaf such that $f(y) \neq g$, it is a witness of the non equivalence between the original formulas. If f is a non empty unate formula, it is non constant and we can set its variables such that $f \neq g$ and prove the non equivalence. On the other hand, if f is non unate, there exists a variable x_j in one term and the same variable appears negated in another one. Then, we keep branching with respect to the values of x_j and again both children have one term less. After at most $|f|$ iterations more f becomes constant and we can solve the problem. The case when f becomes constant but not g is treated in a symmetric way. Thus, our recursion tree has depth at most $\mathcal{O}(\log n)$, the whole tree is polynomial size and we are done. □

This lemma gives us a effective procedure to decide the equivalence between the formulas together with a way to find a witness in the case that they are not equivalent. Thus, we can derive the following theorem:

Theorem 11. *The class of $\mathcal{O}(\sqrt{\log n})$-size CDNF is exact learnable in polynomial time using membership queries and a depth-3 formula as a hypothesis.*

Proof. We just have to observe that Bshouty's algorithms B and B^δ, when learning a formula $f \in \mathcal{O}(\sqrt{\log n})$-term DNF $\cap$ $\mathcal{O}(\sqrt{\log n})$-clause CNF always produce hypothesis of the form $h = h_1 \wedge \ldots \wedge h_r$ and $h^\delta = h_1^\delta \vee \ldots h_s^\delta$ where r is at most the CNF size of f and s is at most the DNF size of f. Moreover, the number of terms in any DNF formula h_i is bounded by the DNF size of f and the number of clauses in any CNF formula h_j^δ is bounded by the CNF size of f. Thus, $|h| + |h^\delta| = \mathcal{O}(\log n)$ and we use Lemma 10 to solve an equivalence query. By Lemma 2 we can exactly learn that class only with membership queries. The running time of both algorithms B and B^δ is polynomial in the formula size since so they are its respective monotone and dual monotone dimension. □

This result is an improvement of the learnability of k-term monotone DNF from membership queries shown in [**BGHM93**] since we are learning a non monotone class and with non constant number of terms. Although it has been proved that k-term monotone DNF for non constant number of terms is not learnable from membership queries only [**BGHM93**], that proof assume that we learn with respect to the DNF size only. Thus, considering the CNF size together with the DNF allows us to overcome that difficulty. Moreover, as we mention before, the number of membership queries needed is optimal [**BC+95**]. Theorem 11 also improves the results in [**BGHM93**] where $\sqrt{\log n}$-term DNF are shown to be exactly learnable with membership and an optimal number of equivalence queries ($\sqrt{\log n}+1$). Again, by considering the CNF size we are able to get rid of the equivalence queries, while considering only the DNF size it is not possible.

5 Conclusions

We have shown that the classes of read-k monotone formulas, Sat-k $\mathcal{O}(\log n)$-DNF and $\mathcal{O}(\sqrt{\log n})$-size CDNF are exactly learnable with only membership queries. All our algorithms use a number of membership queries bounded by $\max\{d(f), c(f)\}$ that matches the lower bounds in [**BC+95**]. Thus, although we are using a measure for the complexity that might seem unrealistic, it is the optimal one for this framework. Moreover,we also propose a systematic way to obtain new learning algorithms from membership queries only. Now, we just have to concentrate in solving the equivalence problem between some other classes of formulas and from there we automatically will derive new learning algorithms. Furthermore, we also show how to solve equivalence for some classes of formulas. These results may be of independent interest outside the learning community. For instance, for the monotone case, solving equivalence has been proved to be equivalent to solve the duality problem (we refer the reader to [**BI94**] for details and further references).

We left as open problem the extension of our results from read-k monotone formulas to read-once and read-twice arbitrary formulas. In this respect, we already have some encouraging results. As we mention before, it is not possible to extend this result to read-thrice formulas since it will imply exact learning of general DNF with membership queries. This is known to be impossible with a polynomial number of membership queries since the class of DNF contains the class of all singletons. It is also open whether the class of Sat-k $\mathcal{O}(\log n)$-DNF or the more general one of $\mathcal{O}(\log n)$-CDNF is properly exact learnable using only membership queries.

6 Acknowledgments

I would like to thank Osamu Watanabe and Ricard Gavaldá for many helpful discussions. I also want to thank Jorge Castro for a simplification of the proof of Theorem 8.

References

[**AHP92**] H. Aizenstein, L. Hellerstein, and L. Pitt. Read-thrice DNF is hard to learn with membership queries and equivalence queries. *Proceedings of the 33rd Annual IEEE Symposium on Foundations of Computer Science*, (1992), 523-532.

[AP91] H. Aizenstein, and L. Pitt. Exact learning of read-twice DNF formulas. *Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science*, (1991), 170-179.

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

[AHK93] D. Angluin, L. Hellerstein, and M. Karpinski. Learning read-once formulas with queries. *Journal of the Association for Computing Machinery*, 40, (1993), 185-210.

[AK91] D. Angluin, and M. Kharitonov. When won't membership queries help? *Proceedings of the 23rd ACM Symposium on the Theory of Computing,* (1991), 444-454.

[BI94] J.C. Bioch, and T. Ibaraki. Complexity of Identification and Dualization of Positive Boolean Functions. *Information and Computation*, 123, (1995), 50-63.

[BF+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. *Proceedings of the 26th ACM Symposium on the Theory of Computing,* (1994), 382-389.

[Bsh95] N.H. Bshouty. Exact Learning Boolean functions via the Monotone Theory. *Information and Computation*, 123, (1195), 146-53.

[BC+95] N.H. Bshouty, R. Cleve, R. Gavaldà, S. Kannan, and C. Tamon. Oracles and Queries that are Sufficient for Exact Learning. *Electronic Colloquium on Complexity.* Technical Report TR95-015, 1995.

[BGHM93] N.H. Bshouty, S.A. Goldman, T. Hancock, and S. Matar. Asking Queries to Minimize Errors. *Proceedings of the 6th Workshop on Computational Learning Theory,* (1993), 41-50..

[Jac94] J. Jackson. An Efficient Membership-Query Algorithm for Learning DNF with Respect to the Uniform Distribution. *Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science*, (1994), 42-53.

[KM93] E. Kushilevitz, and Y. Mansour. Learning decision trees using the Fourier spectrum. *SIAM Journal on Computing*, 22, (1993), 1331-1348.

[MI94b] K. Makino, and T. Ibaraki. The Maximum Latency and Identification of Positive Boolean Functions. *ISAAC'94, Algorithms and Computation*, *Lecture Notes in Computer Science*, 834, (1994), 324-332.

[PR95] K. Pillaipakkamnatt, and V. Raghavan. Read-Twice DNF Formulas are Properly Learnable. *Information and Computation*, 122, (1995), 236-267.

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

This article was processed using the LaTeX macro package with LLNCS style