To appear in *Proc. Twelfth Annual ACM Conference on Computational Learning Theory (COLT 99)*, July, 1999.

On Theory Revision with Queries

Robert H. Sloan*
Dept. of EE & Computer Science
University of Illinois at Chicago
851 S. Morgan St. Rm 1120
Chicago, IL 60607-7053
sloan@eecs.uic.edu

György Turán†
Dept. of Mathematics, Statistics & Computer Science
University of Illinois at Chicago,
Research Group on Artificial Intelligence
Hungarian Academy of Sciences
gyt@uic.edu

Abstract

The theory revision, or concept revision, problem is to correct a given, roughly correct concept. Given the representation of an initial concept, one would like to obtain a representation of the target concept by applying *revisions*, that is, syntactic modifications such as the deletion of a variable or a term. We give efficient revision algorithms using membership and equivalence queries for 2-term monotone DNF, monotone k-DNF, and read-once formulas. An example is given showing that some monotone DNF formulas cannot be revised efficiently. These results all assume that the revisions allowed are the replacements of a variable occurrence with a constant, which, for DNFs, corresponds to deletions of variables and terms. We also discuss a more general error model where besides deletions, additions are also allowed.

1 INTRODUCTION

What the computational learning theory community calls a *concept* is often referred to as a *theory* elsewhere in artificial intelligence and logic. Thus the machine learning problem known as *theory revision* may be viewed as the problem of correcting a given, roughly correct concept.

For instance, one may have somehow acquired a roughly correct theory/concept for the desired target concept "Stock That Will Significantly Appreciate". The theory is (*LowPE* $\wedge$ *HighProfits*) $\vee$ *LowPriceToBookValue* $\vee$ (*InternetRelated* $\wedge$ *NonzeroProfits*). However, Amazon.com (all attributes false except *InternetRelated* true) is a positive counterexample. The theory must be *revised* (or *edited*), perhaps by revising the term (*InternetRelated* $\wedge$ *NonzeroProfits*) to *InternetRelated*.

*Partially supported by NSF grant CCR-9800070.

†Partially supported by NSF grant CCR-9800070, OTKA grant T-025271, and FKFP 1354/97.

We are aware of only one previous work that uses a learning theory approach to the theory revision problem, due to Mooney [20]. His results, which we discuss a bit more later, treat the problem from the point of view of sample complexity within the PAC model. Computational efficiency was posed as an open problem.

A typical application for theory revision is revising a theory such as a set of rules generated by an expert. A common assumption is that the theory comes with a set of labeled examples, which always includes some that disagree with the classification provided by the theory. Typically in the AI literature one is supposed to make a "small" revision to the theory so that it classifies all the given examples correctly.

Here, we formulate this problem in the model of learning with membership and equivalence queries. The use of equivalence queries appears to be reasonable in the scenario just discussed, as each equivalence query can be simulated by cycling through the given set of examples until we find a counterexample to our current version of the theory. If no counterexamples are found, then the learning process terminates. This simulation can never require more queries than the worst-case number in the formal model. In addition, when the initial theory is provided by an expert, it may be realistic to assume that the expert can correctly answer membership queries.

1.1 RELATED WORK

Theory revision is one of several problems, also including *theory restructuring*, which is aimed at making a theory more efficient or transparent, that make up the area of *theory refinement* in machine learning (see, e.g., Wrobel [28, 29]). There are many systems for theory revision in propositional and predicate logic (e.g., [15, 21, 23, 26]). For instance, the EITHER system [21] uses deductive, abductive, and inductive components to revise a propositional Horn theory from a given set of counterexamples.

Formal work in theory revision includes Koppel et al. [15]. They considered the problem of fixing a given theory for a given set of examples, and gave a positive result. Argamon-Engelson and Koppel also considered a version of the revision problem called *theory patching* where there is a prespecified part of the initial theory that cannot be modified [5].

Mooney [20] formulated an approach to the study of theory revision in computational learning theory based on *syntactic distances*. The syntactic distance between a given concept representation and another concept is the minimal number of elementary operations (such as the addition or the deletion of a literal or a clause) needed to transform the given concept representation to a representation of the other concept. Mooney proposed considering the PAC-learnability of the class of concepts having a bounded syntactic distance from a given concept representation. This approach provides an interesting formalization of the availability of *prior knowledge* to the learner, even in propositional logic. Mooney observed that for a fixed number of revisions, the polynomial size of these concept classes implies that only *logarithmic* sample size is needed.

The problem of theory revision appears in several other guises, including *program debugging* and *system diagnosis*. Indeed, one of the motivating examples in Angluin's seminal paper on learning with queries [2] is Shapiro's interactive program debugging system [25], where the user is required to provide the answers to the queries. Reducing the query complexity of Shapiro's algorithm has been of some interest in machine learning (e.g., [1]).

When theories are expressed as propositional Boolean formulas, the theory revision problem is related to *fault analysis of circuits* in switching theory (see, e.g., Kohavi [14]). There a major problem is simply fault *testing*, determining whether the given circuit works correctly. (Faulty circuits may simply be thrown away.)

Model-based diagnosis is a similar but slightly less closely related field. Davis and Hamscher [9] give a good survey of the field through 1987 or so; there is also lots of more recent work (e.g., de Kleer et al. [19] and Marcotte et al. [18]). One of the key concerns of model-based diagnosis is handling a very broad set of *errors* that is typically unknowable in advance. However, for theory revision, while a great variety of errors are considered, it is possible to specify the set of possible errors in advance. In the revision of theories in propositional logic it may be assumed that literals and clauses are *superfluous* and/or *missing*. We view these possibilities as operations available to edit the initial formula, and therefore refer to them as *revision operators*. Many model-based diagnosis systems rely implicitly or explicitly on solving some NP-complete or inherently exponential problem (e.g., enumerating all the minimal hitting sets of a set system [22]).

Several areas of machine learning have at least some connection to theory revision. The study of *drifting concepts* in learning theory [12] may perhaps be viewed as a "dynamic" version of theory revision. Much has been written about *learning from partial information*. A few examples include SOAR [16] in systems-building machine learning, the work of Case et al. [8] and Jain and Sharma [13] in inductive inference, and Rivest and Sloan [24] in PAC learning. Of course, theory revision can be viewed as simply "learning from *a lot* of partial information." (And all machine learning is then the special case of learning from partial information where the partial information is the empty set.) However, in practice, different techniques can be brought to bear when one knows a close approximation to the correct concept instead of just some partial information about it. In this work, we seek to formally model this phenomenon.

1.2 OVERVIEW OF THIS PAPER

We extend Mooney's syntactic distance approach to query models, in particular to learning with equivalence and membership queries. We consider the complexity of concept revision[1] for basic classes such as DNF and read-once formulas. Our revision operator, or, equivalently, error model, allows for the replacement of an *occurrence of a variable* by a constant 0 or 1. This corresponds to allowing the *deletion* of a variable or a clause in a DNF in theory revision [5], and to the "stuck at" faults in circuit fault analysis. Thus a *revision algorithm* for a formula φ has to identify a target concept represented by a formula that can be obtained from φ by replacing certain variable occurrences by constants, using equivalence and membership queries.

A revision algorithm is *efficient* if its query complexity depends polynomially on the syntactic distance between the initial formula and the target, and only polylogarithmically on the size of the concept representation. All the algorithms presented in this paper have running time that is polynomial in the syntactic distance between the initial formula and the target, and in the size of the concept representation.

It is an interesting feature of revision complexity that a single formula, such as a particular DNF, induces a concept class, rather than being a single concept in a concept class. Thus a revision query complexity is associated with each individual formula through the (standard) learning complexity of the concept class of its revisions.

The efficiency criterion is similar to the notion of efficiency for *attribute efficient* learning algorithms [6, 7]. In fact, there seems to be an interesting relationship between concept revision and attribute efficient learning. Informally, in attribute efficient learning, most variables are *irrelevant*, while in concept revision most variables *act in a known way*.

We obtain efficient concept revision algorithms for 2-term unate DNF, unate k-DNF and read-once formulas. The tractability of read-once formulas is a general phenomenon in the areas discussed above. Koppel et al.'s PTR algorithm [15] converges for a closely related class, and read-once formulas (called fanout free circuits in switching theory) provide the efficiently solvable case for some methods of fault diagnosis [14]. One of these methods, called "path sensitization," is used in our learning algorithm as well. We also give an example of a monotone DNF for which there is no efficient concept revision algorithm. As the concept classes considered are small, but very efficient learning algorithms are required, one needs somewhat different techniques than the previous ones [2, 4] for these much studied concept classes. Thus even 2-term monotone DNF, which are very simple for learning with equivalence and membership queries, present difficulties for a revision algorithm.

Besides their query complexity, revision algorithms can also be evaluated by the number of revisions performed. In AI it is typically required to perform as few revisions as possible [11]. In general, this is hard to achieve [10]. Some of

[1]The AI notion of a theory encompasses both the notion of a single concept as well as richer constructs, such as multiple concepts being learned at the same time. Certainly the revision of propositional logic concepts falls within the AI notion of theory revision, and in this paper we will use the terms theory revision and concept revision interchangeably.

our algorithms do perform as few revisions as possible; we will discuss this more in the final paper.

The paper is organized as follows. After the preliminaries of Section 2, we give the DNF revision algorithms in Section 3 and the read-once formula revision algorithm in Section 4. Section 5 contains the example of a hard DNF. Section 6 discusses a more general error model where additions are also allowed and it gives examples showing that some positive results can be obtained in this model as well. Some open problems are mentioned in Section 7.

2 PRELIMINARIES

We use the standard model of membership and proper equivalence queries (with counterexamples), denoted by MQ and EQ [2]. In an equivalence query, the learning algorithm proposes a *hypothesis*, a concept H from the concept class, and the answer depends on whether $H = C$, where C is the target concept. If so, the answer is "correct", and the learning algorithm has succeeded in its goal of exact identification of the target concept. Otherwise, the answer is a *counterexample*, any instance x such that $C(x) \neq H(x)$. In a membership query, the learning algorithm gives an instance x and the answer is either 1 or 0, depending on $C(x)$, that is, $\mathrm{MQ}(x) = C(x)$, where again C is the target concept.

We also use standard notions from propositional logic such as variable, term, disjunctive normal form (DNF), k-term DNF, k-DNF, monotone, unate, cover, etc. Our read-once formulas will be over AND ($\wedge$) and OR ($\vee$) and we assume that all negations occur at the variables. We assume throughout that the concepts TRUE and FALSE are included with monomials, and hence with any sort of DNF. The all 0 vector will be denoted $\mathbf{0}$; the all 1 vector $\mathbf{1}$.

Let φ be a Boolean formula using the variables $x_1, \ldots, x_n$. Then $C_\varphi \subseteq \{0,1\}^n$ is the concept represented by φ, that is, C_φ is the set of satisfying truth assignments for φ. For instance if $\varphi = (x_1 \wedge x_2) \vee (x_1 \wedge x_3)$, then C_φ is $\{110, 101, 111\}$.

With the exception of Section 6, our revision operator is *fixing an occurrence of a variable* in a formula to a constant. For instance, if we fix x_2 in φ to 1, we obtain the revised formula $(x_1 \wedge 1) \vee (x_1 \wedge x_3)$, which can be simplified to $x_1 \vee (x_1 \wedge x_3)$, and is equivalent to x_1. If instead we fix the second occurrence of x_1 to 0, we obtain the revised formula $(x_1 \wedge x_2) \vee (0 \wedge x_3)$, which can be simplified to $x_1 \wedge x_2$. Notice that for DNF our revision operator could alternately be viewed as two operators: *deleting one literal from a term* (by fixing the occurrence of its variable to make the literal true), and *deleting a term* (by fixing one of its variable occurrences to make the literal false).

For read-once formulas, where there is only one occurrence of the variable(s) being fixed, we write a revision using *substitution* notation, $\sigma = (x_i \rightarrow c_i)$, where c_i is a constant. For example, applying the substitution $\sigma = (x_2 \rightarrow 0, x_3 \rightarrow 1)$ to our formula φ gives the revised formula $(x_1 \wedge 0) \vee (x_1 \wedge 1)$, which simplifies to x_1.

We denote by $\mathcal{R}_\varphi$ the set of formulas obtained from φ by fixing some occurrences of some variables to constants. The corresponding concept class is denoted by $\mathcal{C}_\varphi$.

The *revision distance* between a formula φ and some concept $C \in \mathcal{C}_\varphi$ is defined to be the minimum number of applications of a specified set of revision operators to φ needed to obtain a formula for C. As noted previously, our specified set of revision operators is fixing an occurrence of a variable to a constant, with the exception of Section 6. Thus, for example, we showed earlier that the revision distance between $\varphi = (x_1 \wedge x_2) \vee (x_1 \wedge x_3)$ and the concept represented by x_1 is 1.

A *revision algorithm* for a formula φ has access to membership and proper equivalence oracles for an unknown target concept $C \in \mathcal{C}_\varphi$ and must return some representation in $\mathcal{R}_\varphi$ of the target concept. Our goal is to find revision algorithms whose query complexity is polynomial in the revision distance between φ and the target, but at most *polylogarithmic* in the size of φ. In fact, our results provide something stronger than a revision algorithm. The algorithms we give in this paper all revise some class of concept classes. That is, our algorithms are meta-algorithms, as they take any formula φ from a specified class of formulas (e.g., read-once formulas) and then function as a revision algorithm for the concept class $\mathcal{C}_\varphi$. Notice that the choice of revision operator(s) plays a double role. First, it defines the concept class: all things reachable from the specified formula with the revision operator(s). Second, it determines the revision distance, and that gives us a performance metric.

In general, we use $\mathbf{x}$ to refer to a binary vector in $\{0,1\}^n$, and x to refer to the monotone term that is the product of variables corresponding to the 1's in $\mathbf{x}$. Likewise, given a monotone term t, the binary vector $\mathbf{t} \in \{0,1\}^n$ has 1's in the positions corresponding to the variables of t. So, for instance, if $n = 3$ and t is $x_1 x_3$, then $\mathbf{t} = 101$. For monotone terms s and t we use $s \cap t$ for the term that is the product of those variables in both s and t. As an example, $x_1 x_2 \cap x_1 x_3 = x_1$. (Thus $s \cap t$ is different from $s \wedge t$, which is the product of variables occurring in either s or t.) For vectors $\mathbf{x}$ and $\mathbf{y}$ we use $\mathbf{x} \cap \mathbf{y}$ for the vector with 1's in those positions where both $\mathbf{x}$ and $\mathbf{y}$ have a 1. We say a term t is an *implicant* of a formula φ, in symbols $t \rightarrow \varphi$, if for all vectors $\mathbf{x}$ if $t(\mathbf{x}) = 1$, then $\varphi(\mathbf{x}) = 1$. We will make use of the fact that if s, t, and u are monotone terms and $s \rightarrow u$ and $t \rightarrow u$, then $s \cap t \rightarrow u$.

Remark: As the formula φ to be revised is given to the learning algorithm, it follows directly from the definitions that if the occurrences of a variable and its negations are exchanged, then the revision complexity does not change. Therefore, every algorithm for a *monotone* formula can be extended to its *unate* versions. For simplicity, we state most of our algorithms for the monotone case, but they also work for the unate case.

3 MONOTONE DNF EXPRESSIONS

One of the simplest kinds of formulas that we could try to revise is a monotone conjunction, or 1-term monotone DNF. We cannot use the usual algorithm of asking one membership query for each variable, because that requires too many queries. It is easy to see that using membership queries alone, we can be forced to use $n-1$ queries to locate a single variable fixed to 1 in an n-variable conjunction.

However, we can use equivalence queries in a very straightforward way. Assume our initial formula is $t = x_1 x_2 \cdots x_n$. Any negative counterexample tells us that some variable must be fixed to 0, and the target concept is FALSE. Given any positive counterexample $\mathbf{x} \in \{0,1\}^n$, we can update our hypothesis from t to $t \cap x$ (i.e., fix to 1 the variables corresponding to 0's in $\mathbf{x}$). Continuing in this way, we get to the target using at most d equivalence queries, where d is the revision distance between t and the target.

Notice that even for this simple problem, we must use the framework of theory revision to obtain an algorithm using d queries independent of n. A regular query algorithm using membership and equivalence queries needs at least n queries to learn monotone monomials. An $\Omega(n)$ bound follows from the VC-dimension lower bound for query algorithms [17], and we think the precise n bound is also well known in computational learning theory. However, we could not find any written source for it, so we give a proof here.

Proposition 1 *Any algorithm using membership and equivalence queries to learn monotone monomials on n variables can be forced to make at least n queries.*

Proof. We give a strategy for an adversary responding to the queries. The adversary initializes two sets of variables I and O to be the empty set. The sets will always be disjoint; I contains variables in the target monomial and O contains variables not in the target monomial. It will always be consistent for the target concept to be any monomial containing all variables in I and no variables from O. The adversary adds at most one variable to $I \cup O$ when he responds to a query. Hence a query bound of n follows.

In response to EQ(FALSE), the adversary gives the positive counterexample $\mathbf{1}$. For EQ(t), where t is a monomial, the adversary first checks whether t is missing any variable $v \in I$, and if so, returns as a negative counterexample the vector that is 0 for v and 1 everywhere else. If t contains any variable $v \in O$, then the response is the positive counterexample vector that is 0 for v and 1 every place else. The next case is when t contains a proper superset of I, but no variables from O. Now the adversary picks a variable v in t but not in $O \cup I$, puts $O = O \cup \{v\}$, and again gives as a positive counterexample the vector that is 0 for v and 1 every place else. The final case is when t contains exactly the variables in I. Now the adversary picks a variable $v \notin O \cup I$, and returns as a negative counterexample the vector that is 1 every place in I and 0 elsewhere, and sets $I = I \cup \{v\}$.

There is one additional special case. In response to EQ(TRUE) when $I = \emptyset$, the negative counterexample $\mathbf{0}$ is given. In this case, the adversary must remember to keep $|O| \leq n - 1$.

For MQ($\mathbf{x}$), if $\mathbf{x}$ is missing a 1 in any of the places specified in the set I, then MQ($\mathbf{x}$) $= 0$. Otherwise, the adversary considers the variables outside $I \cup O$. If they are all set to 1, then MQ($\mathbf{x}$) $= 1$. Else, if $v \notin I \cup O$ is set to 0, then MQ($\mathbf{x}$) $= 0$ and the adversary sets $I = I \cup \{v\}$.

This strategy forces the learner to make at least n queries, until $|I| + |O| = n$ (or until $|O| = n - 1$, and the additional query EQ(TRUE) with $I = \emptyset$.).[2] □

[2] If TRUE is not considered to be in the concept class of monomials, then one gets a bound of $n - 1$, because if $|O| = n - 1$ then the remaining variable must be the monomial (i.e., $I \neq \emptyset$).

3.1 A REVISION ALGORITHM FOR MONOTONE 2-TERM DNF EXPRESSIONS

Suppose we are given the initial DNF expression $\varphi_1 = t_1 \vee t_2 = uvwx \vee xyz$. An obvious strategy is to start with the query EQ(φ_1). Assume we receive back the positive counterexample $\mathbf{x} = 011110$. Next make a membership query for each term, that is, for $\mathbf{x} \cap \mathbf{t}_1$ and $\mathbf{x} \cap \mathbf{t}_2$. Suppose the response is MQ$(011100) = 1$ and MQ$(000110) = 0$. We then revise our hypothesis to $\varphi_1' = vwx \vee xyz$. Using a constant number of queries, we "made progress" in the sense that we have presumably reduced the revision distance between our hypothesis and the target.

However, difficulties can arise. Suppose we start with the initial DNF expression $\varphi_2 = wyz \vee xyz$ and EQ(φ_2) returns the positive counterexample 0011. This time there is no point in making membership queries for each term, since this counterexample has the same intersection with each term. We might decide our new hypothesis is yz, but of course the true target could have two terms, and it is unclear how we will determine what the second term should look like.

Another potential difficulty arises from the initial formula $\varphi_3 = x_1 x_2 \cdots x_{n-1} \vee x_{n-1} x_n$. Suppose the target concept is actually x_{n-1}. The revision distance between the two concepts is only 1: delete the literal x_n from the second term, that is, fix x_n to 1. We must be careful that the adversary answering our queries does not force us instead to approach the target by deleting $n - 2$ literals from the first term, using $\Omega(n)$ queries.

Our learning algorithm repeatedly uses a subroutine REFINEHYP, which takes a hypothesis that approximates the target from below, and gets a positive counterexample. If REFINEHYP succeeds, then it acts a little like the "Reduce" subroutine from the general "use equivalence queries to get a new term, and then membership queries to reduce that term" strategy used for query learning of monotone DNF [2]. If REFINEHYP fails, then we need more complicated arguments to proceed.

REFINEHYP(h)
  repeat forever
    if ($\mathbf{x}$ = EQ(h)) == "correct"
      then exit everything. Report success with h.
    for each term t in h
      if MQ($\mathbf{t} \cap \mathbf{x}$) == 1
        then edit t (in h) to $t \cap x$
    if all MQs were 0
      then return example $\mathbf{x}$ and hypothesis h

Figure 1: Algorithm REFINEHYP to refine a hypothesis by deleting variables from the current terms.

Before we describe the performance guarantee of Algorithm REFINEHYP, we need to introduce a definition. We say that DNF $h = \bigvee_{i=1}^{\ell} t_i$ has a *one-to-one mapping into* a target DNF if the target DNF contains ℓ distinct terms $t_1^*, \ldots, t_\ell^*$ such that $t_i \to t_i^*$ for $1 \leq i \leq \ell$. Observe that if h has a one-to-one mapping into the target, then it is a subset of the target (viewing both as subsets of $\{0,1\}^n$). Therefore any counterexample to h must be positive.

Lemma 2 *Let hypothesis h be a monotone DNF, $h = \bigvee_{i=1}^{\ell} t_i$ such that $\mathrm{MQ}(\mathbf{t}_i) = 1$ for $i = 1, \dots, \ell$ and $\mathrm{MQ}(\mathbf{t}_i \cap \mathbf{t}_j) = 0$ for all $i \neq j$. Assume that h has a one-to-one mapping into the target DNF. The following claims hold for* REFINEHYP(h)*:*

1. *Each iteration of the main* **repeat** *loop uses one equivalence query and ℓ membership queries, and each iteration of the main* **repeat** *loop except the last one deletes at least one literal from one term of h.*
2. *The invariant that $\mathrm{MQ}(\mathbf{t}_i \cap \mathbf{t}_j) = 0$ for all $i \neq j$ is maintained.*
3. *The invariant that h has some one-to-one mapping into the target DNF is maintained.*
4. *When* REFINEHYP *returns a counterexample $\mathbf{x}$, then for each term t in h it holds that $\mathrm{MQ}(\mathbf{t} \cap \mathbf{x}) = 0$. Also, $\mathbf{x}$ must be a positive counterexample, so there is a term in the target concept that covers $\mathbf{x}$.*
5. *If the target concept can be represented by ℓ terms, then* REFINEHYP *terminates by reporting success after making at most $O(\ell d)$ queries, where d is the revision distance between h and the target concept.*

Proof. The first three items are proved by induction on the number of edits. The query usage and deletions specified by the first item are obvious, given that Item 3 ensures that all counterexamples are positive counterexamples.

Item 2: Immediately before we edit t_i to become $t_i \cap x$, we know that $\mathrm{MQ}(\mathbf{t}_i \cap \mathbf{t}_j) = 0$ for every $j \neq i$. It must still be true for the edited term $(t_i \cap x)$ that $\mathrm{MQ}((\mathbf{t}_i \cap \mathbf{x}) \cap \mathbf{t}_j) = 0$, since the 1's in the vector $(\mathbf{t}_i \cap \mathbf{x}) \cap \mathbf{t}_j$ are a subset of the 1's in the vector $\mathbf{t}_i \cap \mathbf{t}_j$.

Item 3: Immediately before we edit t_i to become $t_i \cap x$, we know that there is a one-to-one mapping of h into the target DNF. In particular, for every term t_j of h there is a distinct t_j^* in the target such that $t_j \to t_j^*$. Now, since $\mathrm{MQ}(\mathbf{t}_i \cap \mathbf{x}) = 1$, we know that $t_i \cap x$ is an implicant of the target DNF, and thus an implicant of some term of the target DNF. For all $i \neq j$, it must be that $\mathrm{MQ}(\mathbf{t}_i \cap \mathbf{x} \cap \mathbf{t}_j) = 0$, since by Item 2 $\mathrm{MQ}(\mathbf{t}_i \cap \mathbf{t}_j) = 0$. Hence $t_i \cap x$ is *not* an implicant of any of the t_j^* with $j \neq i$, and so it is an implicant of a distinct term of the target. Thus we still have some one-to-one mapping into the target.

Item 4: The fact that $\mathrm{MQ}(\mathbf{t} \cap \mathbf{x}) = 0$ for every t in h is guaranteed by the membership queries made during the last iteration of the **repeat** loop. Also, h has a one-to-one mapping into the target, so as noted before, $\mathbf{x}$ must be a positive counterexample, and therefore some term in the target concept must cover it.

Item 5: If the target concept can be represented by an ℓ-term DNF, then by Item 3, our hypothesis contains an implicant for every term in the target DNF. Now Item 4 guarantees that we never terminate by the **return** statement, because for any $\mathbf{x}$ covered by a term of the target concept, there must be some term t in h such that $\mathrm{MQ}(\mathbf{t} \cap \mathbf{x}) = 1$.

Let the target concept be $\bigvee_{i=1}^{\ell} t_i^*$. It must be that the correspondence of t_i's to t_j^*'s given by the one-to-one mapping is unique and unchanged during the algorithm. If $t_i \to t_i^*$ and also $t_j \to t_i^*$, then $t_i \cap t_j \to t_i^*$, contradicting the fact that $\mathrm{MQ}(\mathbf{t}_i \cap \mathbf{t}_j) = 0$. Therefore there is a unique set of deletions that takes us from the initial formula to the target formula, and every deletion performed by the algorithm is one of those deletions. Algorithm REFINEHYP makes at most ℓ membership queries and one equivalence query for each deletion, giving the $O(\ell d)$ query bound. □

Theorem 3 *Every monotone 2-term DNF φ has a revision algorithm that uses $O(d)$ queries, where d is the revision distance between φ and the target concept.*

Proof. Our general strategy is to approximate the target concept from below. That is, every equivalence query we make must return a positive counterexample (or "correct"). Let $\varphi = t_1 \vee t_2$. Our algorithm begins with membership queries on $\mathbf{t}_1$ and $\mathbf{t}_2$. If both return 0, then the answer is the concept FALSE.

If one query returns 0 and one returns 1, then we know that the target can be represented by a single term. *W.l.o.g.*, say $\mathrm{MQ}(\mathbf{t}_1) = 1$ and $\mathrm{MQ}(\mathbf{t}_2) = 0$. Then call REFINEHYP$(t_1)$. By Lemma 2, Item 5 this must return the correct hypothesis in $O(d)$ queries.

Otherwise, $\mathrm{MQ}(\mathbf{t}_1) = \mathrm{MQ}(\mathbf{t}_2) = 1$. Now, make the membership query $\mathrm{MQ}(\mathbf{t}_1 \cap \mathbf{t}_2)$. If it returns 0, then the hypothesis $t_1 \vee t_2$ satisfies all the conditions of Lemma 2. In particular, it is a two term hypothesis and the target concept must be represented by a two term DNF. So, by Item 5 of Lemma 2, REFINEHYP$(t_1 \vee t_2)$ terminates with a correct hypothesis after $O(2d) = O(d)$ queries.

If $\mathrm{MQ}(\mathbf{t}_1 \cap \mathbf{t}_2) = 1$, then call REFINEHYP$(t_1 \cap t_2)$. If the target concept can be represented by one term alone, then as before, we are done by Lemma 2.

Otherwise, it must be that the target concept must be represented by two terms, that is, it is of the form $t_a^* \vee t_b^*$. Then by Lemma 2, REFINEHYP eventually returns a single term h and $\mathbf{x}$ such that *w.l.o.g.* $h \to t_a^*$, and $h(\mathbf{x}) = t_a^*(\mathbf{x}) = 0$, and $t_b^*(\mathbf{x}) = 1$. Furthermore, the term h has been edited by an amount proportional to the number of queries made by REFINEHYP.

If t and t' are terms such that $t \to t'$, that is, the set of variables of t' is a subset of the set of variables of t, then we write $|t \setminus t'|$ for the number of variables that occur in t but not in t'. With this notation, the number of queries made so far is $|t_1 \cap t_2 \setminus h| + O(1)$.

Next we make the two membership queries $\mathrm{MQ}(\mathbf{t}_1 \cap \mathbf{x})$ and $\mathrm{MQ}(\mathbf{t}_2 \cap \mathbf{x})$. At least one must return 1, because $\mathbf{x}$ is a positive instance. So there are two possibilities.

Case I: Exactly one membership query returned 1; say $\mathrm{MQ}(\mathbf{t}_1 \cap \mathbf{x}) = 1$. Then call REFINEHYP$(h \vee (t_1 \cap x))$. Observe that the conditions for Lemma 2 to apply to this call of REFINEHYP are met. First, $h \to t_a^*$. Second, $x \not\to t_a^*$ (since $t_a^*(\mathbf{x}) = 0$), so $t_1 \cap x \not\to t_a^*$. Thus it must be that $t_1 \cap x \to t_b^*$. Also, $\mathrm{MQ}(\mathbf{h} \cap \mathbf{t}_1 \cap \mathbf{x}) = 0$ because $\mathrm{MQ}(\mathbf{h} \cap \mathbf{x}) = 0$ by Item 4 of Lemma 2 applied to the previous call of REFINEHYP. So, since the target concept has two terms, Lemma 2 Item 5 guarantees that this call of REFINEHYP will terminate with the correct hypothesis using

$$O\left(|h \setminus t_a^*| + |t_1 \cap x \setminus t_b^*|\right)$$

queries. Thus the total number of queries is

$$O\left(|t_1 \cap t_2 \setminus h| + |h \setminus t_a^*| + |t_1 \cap x \setminus t_b^*|\right). \qquad (1)$$

This counting includes the additional $O(1)$ queries asked at the beginning of the algorithm.

Now, notice that there is a unique correspondence between the initial terms and the final terms. In particular, $t_2 \not\rightarrow t_b^*$, since $x \rightarrow t_b^*$ and $\mathrm{MQ}(\mathbf{t}_2 \cap \mathbf{x}) = 0$. Thus t_2 must be revised to t_a^* (and hence t_1 must be revised to t_b^*). So the revision distance between the initial formula and the target is

$$\begin{aligned}|t_1 \setminus t_b^*| + |t_2 \setminus t_a^*| &\geq |t_1 \cap x \setminus t_b^*| + |t_1 \cap t_2 \setminus t_a^*| \\ &= |t_1 \cap x \setminus t_b^*| + |t_1 \cap t_2 \setminus h| + |h \setminus t_a^*|.\end{aligned}$$

The query bound then follows by comparing this to Equation 1.

Case II: Both membership queries returned 1; that is, $\mathrm{MQ}(\mathbf{t}_1 \cap \mathbf{x}) = \mathrm{MQ}(\mathbf{t}_2 \cap \mathbf{x}) = 1$. We show that in this case it must be that $\mathrm{MQ}(\mathbf{t}_1 \cap \mathbf{t}_2 \cap \mathbf{x}) = 1$. It follows from Item 4 of Lemma 2 that neither $t_1 \cap x$ nor $t_2 \cap x$ is an implicant of t_a^*. We have $(t_1 \cap t_2) \rightarrow h \rightarrow t_a^*$ and thus also $h \cap t_1 \cap t_2 = h$. If $t_1 \cap x \rightarrow t_a^*$, then $(t_1 \cap x) \cap (t_1 \cap t_2) = (t_1 \cap t_2 \cap x) \rightarrow t_a^*$. In that case, $h \cap t_1 \cap t_2 \cap x = h \cap x \rightarrow t_a^*$, which we know is false. Thus $t_1 \cap x \not\rightarrow t_a^*$, and similarly $t_2 \cap x \not\rightarrow t_a^*$. So it must be that both are implicants of t_b^*; thus their intersection is also an implicant of t_b^*. Furthermore, $\mathrm{MQ}(\mathbf{h} \cap \mathbf{t}_1 \cap \mathbf{t}_2 \cap \mathbf{x}) = \mathrm{MQ}(\mathbf{h} \cap \mathbf{x}) = 0$. So we can call REFINEHYP$(h \vee (t_1 \cap t_2 \cap x))$. Now there is no longer a unique mapping from original terms to final terms, as both final terms are contained in $t_1 \cap t_2$. However, the revision distance is correctly computed by considering either of the two mappings. With this observation, the argument is similar to Case I. In particular, the total number of queries made is

$$O\left(|t_1 \cap t_2 \setminus h| + |h \setminus t_a^*| + |t_1 \cap t_2 \cap x \setminus t_b^*|\right),$$

and the revision distance between the initial formula and the target is

$$\begin{aligned}|t_1 \setminus t_b^*| + |t_2 \setminus t_a^*| &\geq |t_1 \cap t_2 \cap x \setminus t_b^*| + |t_1 \cap t_2 \setminus t_a^*| \\ &= |t_1 \cap t_2 \cap x \setminus t_b^*| + |t_1 \cap t_2 \setminus h| + |h \setminus t_a^*|,\end{aligned}$$

and the query bound again follows by comparing these two quantities. □

3.2 A REVISION ALGORITHM FOR MONOTONE k-DNF EXPRESSIONS

Now we consider the problem of revising a monotone k-DNF formula.

Lemma 4 *Let $\mathbf{x}$ be a satisfying truth assignment for an unknown monotone k-DNF ψ. Then one can identify a term of ψ satisfied by $\mathbf{x}$, using $O(k \log n)$ membership queries to ψ.*

Proof. We give an informal description of a procedure FINDMINTERM$(\psi, \mathbf{x})$ that performs this task. An initial membership query $\mathrm{MQ}(\mathbf{0})$ decides if ψ is TRUE. Let us assume that this is not the case. For every term t let us consider the sequence of the indices of variables occurring in t in decreasing order. Let t^*, corresponding to the sequence $i_1 > \cdots > i_\ell$, $\ell \leq k$, be the lexicographically first term in ψ that covers $\mathbf{x}$. [3] We claim that this term can be found using $O(k \log n)$ membership queries to ψ.

Let $\mathbf{x}_i$, $0 \leq i \leq n$, be the vector obtained from $\mathbf{x}$ by leaving it unchanged on the first i positions, and setting all its other bits to 0. Then i_1 is the smallest number i such that $\mathrm{MQ}(\mathbf{x}_i) = 1$. It can be found using binary search on the sequence $\mathbf{0} = \mathbf{x}_0 \leq \mathbf{x}_1 \leq \cdots \leq \mathbf{x}_n = \mathbf{x}$, and it holds that $i_1 > 0$. Now ask the membership query $\mathrm{MQ}(\mathbf{e}_{i_1})$, where $\mathbf{e}_{i_1}$ has a single 1 component, at the i_1th position. If the answer is 1, then $t^* = x_{i_1}$. Otherwise, let $\mathbf{x}_i^{i_1}$, $0 \leq i < i_1$ be the vector obtained from $\mathbf{x}$ by keeping it unchanged on the first i positions and on the i_1th position, and turning all its other bits off. Then i_2 is the smallest number i such that $\mathrm{MQ}(\mathbf{x}_i^{i_1}) = 1$ and again it can be found by binary search. Continuing this process, t^* is identified using $O(k \log n)$ membership queries. □

Theorem 5 *Every k-DNF formula φ on n variables has a revision algorithm that uses $O(kd \log n)$ queries, where d is the revision distance between φ and the target concept.*

```
REVISEkDNF(φ)
  h = φ
  while (x = EQ(h)) ≠ "correct"
    if x is a negative counterexample
      delete all terms covering x from h
    else
      t* = FINDMINTERM(ψ, x)
      h = h ∨ t*
      Delete from h every proper implicant of t*
```

Figure 2: Revision algorithm for k-DNF formulas

Proof. The query algorithm REVISEkDNF is shown in Figure 2. We start with φ as our initial hypothesis, and repeatedly make equivalence queries with our current hypothesis h.

Whenever we receive a negative counterexample $\mathbf{x}$, we delete from our hypothesis those terms that cover $\mathbf{x}$. Notice that the deleted terms *must* be deleted in any editing to reach the target, so when we get negative counterexamples we "spend" only one query for at least a gain of one in revision distance. Also notice that the deleted terms must be original terms of φ, not terms added by FINDMINTERM, because all terms added by FINDMINTERM are correct.

Now let us assume that we receive a positive counterexample $\mathbf{x}$. Then $\mathbf{x}$ satisfies a term in the target that is obtained from some term in φ by deleting at least one variable. We run the procedure FINDMINTERM to find such a term t^*, and then we add t^* as a new term of h.

Notice that right after setting h to $h \vee t^*$, our hypothesis h will generally not be as compact as possible and sometimes will not belong to $\mathcal{R}_\varphi$. (E.g., with $n = 3$, we might update h from $xy \vee xz$ to $xy \vee xz \vee x$, which has more terms than the original formula.) Therefore, we need to transform h to a semantically equivalent formula that is in $\mathcal{R}_\varphi$, by removing all proper implicants of t^* from h. Below we argue that the output h is indeed in $\mathcal{R}_\varphi$.

We now finish our analysis of the query complexity of REVISEkDNF, begun with the discussion of negative counterexamples. Each call to FINDMINTERM finds a distinct new minterm of the target that was not contained in φ. At

[3] In this ordering, a set precedes another if at the first position where they differ it has a smaller number (or none). Thus, e.g., $\{10, 7\}$ precedes $\{10, 7, 3\}$, which precedes $\{10, 7, 4, 2\}$.

least one edit is required to obtain each such minterm, so the number of calls to FINDMINTERM is at most the revision distance between φ and the target. The total query bound then follows from Lemma 4.

Now we must argue that there is a revision that edits φ to the h produced at the end of the main **while** loop.

First we introduce some terminology. Let φ be of the form $t_1 \vee \cdots \vee t_\ell$, and let ψ be a representation of the target. Assume that a monotone DNF g is of the form $t_1 \vee \cdots \vee t_m \vee t_1^* \vee \cdots \vee t_r^*$, for $m \leq \ell$ such that the following properties hold:

- every t_j^* is a minterm of ψ but not an implicant of φ
- every t_i for $m < i \leq \ell$ (i.e., the terms of φ missing from g), either covers a negative instance of ψ, or is a proper implicant of some t_j^*
- no t_i for $1 \leq i \leq m$ (i.e., the terms of φ present in g), is a proper implicant of any t_j^*.

We claim that under these conditions $g \in \mathcal{R}_\varphi$ and we can find a suitable revision in polynomial time (without using any queries). This follows by considering the bipartite graph that has the terms of φ as its left vertices and the terms of g as its right vertices, with edges corresponding to the implicant relation, and showing that this graph has a matching that matches every right vertex.

It can be shown by induction on the iteration of the **while** loop of REVISEkDNF that h satisfies the above conditions for g. □

4 A REVISION ALGORITHM FOR READ-ONCE FORMULAS

In this section we give an efficient revision algorithm for read-once formulas. The algorithm uses the technique of *path sensitization* from fault analysis in switching theory (see, e.g., [14]). Let us assume that we would like to revise the monotone read-once formula $(\varphi_1 \vee \varphi_2) \wedge \varphi_3$. Thus φ_1, φ_2, and φ_3 are monotone read-once formulas over disjoint sets of variables. Let the target formula be $(\psi_1 \vee \psi_2) \wedge \psi_3$, where $(\psi_1 \vee \psi_2) \wedge \psi_3$ is obtained from $(\varphi_1 \vee \varphi_2) \wedge \varphi_3$ by replacing certain variables by constants. Fix all the variables in φ_2 to 0, and all the variables in φ_3 to 1. Form two vectors $\mathbf{a}_0$ and $\mathbf{a}_1$ by fixing the remaining variables to 0, resp., to 1, and ask the membership queries $\mathrm{MQ}(\mathbf{a}_0)$ and $\mathrm{MQ}(\mathbf{a}_1)$. There are three possibilities. If $\mathrm{MQ}(\mathbf{a}_1) = 0$, then it must be the case that *either* $\psi_1(\mathbf{1}) = 0$, in which case ψ_1 is identically 0, *or* $\psi_3(\mathbf{1}) = 0$, in which case the whole target formula is identically 0. If $\mathrm{MQ}(\mathbf{a}_0) = 1$, then it must be the case that *either* $\psi_1(\mathbf{0}) = 1$, in which case ψ_1 is identically 1, *or* $\psi_2(\mathbf{0}) = 1$, in which case ψ_2 is identically 1. The fixing of the variables of φ_2 to 0 and those of φ_3 to 1 is called *sensitizing* φ_1, and the corresponding partial truth assignment α is called the *sensitizing truth assignment* for φ_1. For the learning algorithm it is also important to notice that we can also gain some information in the remaining third case, when $\mathrm{MQ}(\mathbf{a}_0) = 0$ and $\mathrm{MQ}(\mathbf{a}_1) = 1$. In this case we do not observe any "abnormality," but we can conclude that for every truth assignment $\mathbf{y}$ to the variables of ψ_1 it holds that $\varphi_1(\mathbf{y})$ is equal to the answer given to the membership query for the vector consisting of $\mathbf{y}$ and α. Thus we can simulate membership queries to the subformula ψ_1 by membership queries to the target concept, and this enables the learning algorithm to proceed by recursion in the third case. Also note that in this case it is still possible that $\psi_2(\mathbf{1}) = 0$ and/or $\psi_3(\mathbf{0}) = 1$.

Here is a detailed example showing how the read-once formula revision algorithm works. Let the formula to be revised be

$$\varphi = ((x_1 \wedge x_2) \vee (x_3 \wedge x_4)) \wedge ((((x_5 \wedge x_6) \vee x_7) \wedge x_8) \vee x_9)$$

and the substitution giving the target formula be

$$\sigma = (x_3 \to 1,\ x_5, x_6, x_8 \to 0). \qquad (2)$$

Thus the target concept is represented by the formula

$$\psi = ((x_1 \wedge x_2) \vee (1 \wedge x_4)) \wedge ((((0 \wedge 0) \vee x_7) \wedge 0) \vee x_9).$$

We start by asking an equivalence query to φ. For the moment, the equivalence query serves only to decide if we have already found the target formula. Depending on the responses to certain membership queries that we ask next, we may use $\mathbf{x}$ shortly. Let us assume that we receive the negative counterexample $\mathbf{x} = 110011110$. Our goal is to find, using a kind of binary search on subformulas, a substitution for some variables, that either occur in σ or are equivalent in effect to some substitutions in σ. For this purpose, we find a subformula of φ that contains about half of the variables, for example

$$\varphi' = (x_1 \wedge x_2) \vee (x_3 \wedge x_4).$$

Let $\psi' = (x_1 \wedge x_2) \vee (1 \wedge x_4)$ be the corresponding subformula of the target. Thus ψ' is obtained from φ' by applying the substitution σ.

Now we form the sensitizing truth assignment α for φ', which in this case simply sets all variables not in φ' to 1, and we ask membership queries for $(\mathbf{0}, \alpha)$ and for $(\mathbf{1}, \alpha)$. The answer is $\mathrm{MQ}(\mathbf{0}, \alpha) = 0$ and $\mathrm{MQ}(\mathbf{1}, \alpha) = 1$, which corresponds to the third case in the discussion at the beginning of this section. Let $\mathbf{x}_1 = 1100$ denote the part of the counterexample $\mathbf{x}$ that belongs to φ' and let $\mathbf{x}_2 = 11110$ denote the rest of $\mathbf{x}$. Thus we can write $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2)$. As explained above, by asking the membership query $\mathrm{MQ}(\mathbf{x}_1, \alpha)$ we can determine that $\psi'(\mathbf{x}_1) = 1$. Knowing φ, we can determine without asking any queries that $\varphi'(\mathbf{x}_1) = 1$. As $\psi'(\mathbf{x}_1) = \varphi'(\mathbf{x}_1)$, it follows that the $\mathbf{x}_2$ part of the counterexample must be responsible for the disagreement between φ and ψ. In this particular case, the variables in $\mathbf{x}_2$ happen to induce a subformula of φ, and so we can continue without further difficulties, whereas in general we will need to call FINDFORMULA at this point. Now, however, we can simply substitute 1 for φ', and then $\mathbf{x}_2 = 11110$ is still a negative counterexample for the new target obtained by applying σ to the new initial formula

$$\varphi_{\mathrm{new}} = 1 \wedge ((((x_5 \wedge x_6) \vee x_7) \wedge x_8) \vee x_9).$$

It is important to note that as we defined $\varphi_{\mathrm{new}}(\mathbf{y}) = \varphi(\mathbf{x}_1, \mathbf{y})$, we can simulate membership queries to the new target by membership queries to the original target; thus we can continue the same procedure recursively.

As the subsequent iterations illustrate additional cases, we give further steps of the algorithm on the example. The next half size subformula is $x_5 \wedge x_6$. The sensitizing truth assignment for this subformula is 010. Now, the membership queries to $(00, 010)$ and $(11, 010)$ both return 0, indicating that either $x_5 \wedge x_6$ or some subformula containing $x_5 \wedge x_6$ is turned into the constant 0. This corresponds to the first case in the discussion at the beginning of this section. By asking additional membership queries we find that $\mathrm{MQ}(11, 110) = 0$ but $\mathrm{MQ}(11, 111) = 1$. This shows that

$$(((x_5 \wedge x_6) \vee x_7) \wedge x_8)$$

is a maximal constant 0 subformula in the target concept. We note that in this case we do not continue recursively, and in this iteration we did not even use the counterexample. The next step is to compute the minimal number of variables that need to be turned to 0 in order to make the subformula identically 0. This can be achieved by the single substitution $x_8 \to 0$. Now we have completed one call of the procedure FINDCONSTANT.

Our current hypothesis is the formula obtained by the substitution just found, that is,

$$h = ((x_1 \wedge x_2) \vee (x_3 \wedge x_4)) \wedge x_9.$$

Let us assume that we receive the positive counterexample 000111111, which, restricted to the five variables in h, is 00011. We continue with the half size subformula $x_1 \wedge x_2$, which divides the counterexample into 00 and 011. The sensitizing partial truth assignment to the second half is 001. We find that $\mathrm{MQ}(00, 001) = 0$ and $\mathrm{MQ}(11, 001) = 1$, thus $(x_1 \wedge x_2)$ is not turned into a constant subformula. (Notice that our only membership oracle needs inputs from $\{0, 1\}^9$; fortunately, we may give any values to the "missing" variables.) The membership query $\mathrm{MQ}(00, 001) = 0$ tells us that the first half of the counterexample gives the same output in $(x_1 \wedge x_2)$ and in the corresponding subformula of the target. To recurse, we must find a *subformula* of h that contains some constant subformula, but the three variables x_3, x_4, and x_9 do not induce a subformula of h.

In this case we need consider only the two subformulas $x_3 \wedge x_4$ and x_9, though in general there could be $\Omega(n)$ such subformulas, necessitating a kind of binary search. By definition, h disagrees with the target on the counterexample, and we have just concluded that $x_1 \wedge x_2$ agrees with the counterexample. So, if subformula $(x_1 \wedge x_2) \vee (x_3 \wedge x_4)$ of h disagrees with the corresponding subformula of the target, then the subformula containing a constant subformula must be $x_3 \wedge x_4$. Otherwise it is x_9. To test whether the subformula $(x_1 \wedge x_2) \vee (x_3 \wedge x_4)$ agrees with the target on the counterexample, we ask a membership query on an instance formed by setting x_1, x_2, x_3, and x_4 to the values they have in the counterexample, and setting the remaining variable (x_9) to the value it had in the sensitizing assignment for $x_1 \wedge x_2$. That, is we make the query $\mathrm{MQ}(00011) = 1$. Since $h(00011) = 0$, which disagrees with the target, there must be a constant subformula in $x_3 \wedge x_4$, which is the input subformula for the next call to FINDCONSTANT.

That call will identify the substitution $x_3 \to 1$, and the next equivalence query to the formula

$$((x_1 \wedge x_2) \vee x_4) \wedge x_9$$

will finally identify the target concept. Notice that we have actually revised *fewer* variables than given in Equation 2. The number of variables revised is as small as possible for obtaining the target concept.

Theorem 6 *Every n-variable read-once formula φ has a revision algorithm that uses $O(d \log^2 n)$ queries, where d is the revision distance between φ and the target concept.*

Proof. It may be assumed *w.l.o.g.* that φ is monotone, as every read-once formula is unate, and as noted at the end of Section 2, the monotone and unate cases are equivalent.

Given a subformula φ' of φ, every truth assignment $\mathbf{x}$ can be written as $(\mathbf{x_1}, \mathbf{x_2})$, where $\mathbf{x_1}$ contains all the variables in φ', and $\mathbf{x_2}$ contains all the variables not in φ'. This is referred to as the φ'*-partition* of $\mathbf{x}$.

An *approximately half size subformula* of a read-once formula contains at least one-third and at most two-thirds of the variables. It is a standard fact that such a subformula exists (see, e.g., [27]). For example, any minimal subformula that contains at least one-third of the variables has this property.

Now we formalize the definition of a sensitizing partial truth assignment that we mentioned previously. Let φ' be a subformula of φ. Consider the binary tree representing φ, and let P be the path leading from the root of φ to the root of φ'. Then φ can be written as

$$(\cdots(\varphi' \circ_r \varphi_r) \circ_{r-1} \cdots \circ_3 \varphi_3) \circ_2 \varphi_2) \circ_1 \varphi_1, \qquad (3)$$

where $\varphi_1, \ldots, \varphi_r$ are the subformulas corresponding to the siblings of the nodes of P, and $\circ_1, \ldots, \circ_r$ are either $\wedge$ or $\vee$. In this representation, we used the commutativity of AND and OR; in general φ' need not be a leftmost subformula of φ.

Let the sets of variables occurring in φ_i be X_i, and the set of variables occurring in φ' be Y. Thus, these sets form a partition of $\{x_1, \ldots, x_n\}$. Now let α be the partial truth assignment that assigns 1 (resp., 0) to every variable in X_i if $\circ_i$ is AND (resp., OR), for every $i = 1, \ldots, r$. Then α is called the *partial truth assignment sensitizing* φ'.

Given a substitution σ, let $\varphi\sigma$ be the formula obtained by replacing each variable in φ by the corresponding constant from σ. A subformula is *constant* if it computes a constant function. Maximal constant subformulas must be pairwise disjoint. Two substitutions σ_1 and σ_2 are *equivalent* if $\varphi\sigma_1$ and $\varphi\sigma_2$ compute the same Boolean function.

The following lemma, whose proof is omitted, is useful because the learning algorithm proceeds by finding maximal constant subformulas.

Lemma 7 *Substitutions σ_1 and σ_2 are equivalent if and only if their sets of maximal constant subformulas are identical.*

The learning algorithm is based on the recursive procedure FINDCONSTANT of Figure 3, which takes a formula φ and a counterexample $\mathbf{x}$ and returns a substitution σ. This substitution fixes a subformula to a constant c, and this subformula must compute constant c in any representation of the target concept.[4] FINDCONSTANT uses membership queries

[4]In several places in the proof we will say that a property holds for any representation of the target concept. Notice that this must be true, as all the information that our algorithm will use comes from membership and equivalence queries about the target, and the responses to such queries are independent of the particular representation.

```
FINDCONSTANT(φ, x)
    if φ has one variable
        return substitution σ fixing it to constant 1 − x
    if MQ(0) == 1 or MQ(1) == 0
        return substitution σ fixing φ to the appropriate constant
    φ' = an approximately half-size formula of φ
    α = the partial truth assignment sensitizing φ'
    if (MQ(0, α) == MQ(1, α) == c)
        then return GROWFORMULA(φ, φ', c)
    else
        (x_1, x_2) = the φ'-partition of x
        if MQ(x_1, α) ≠ φ'(x_1)
            then FINDCONSTANT(φ(·, α), x_1)          // look in φ'
            else
                i = FINDFORMULA(φ, φ', x)
                FINDCONSTANT(φ_i, x_{2,i})           // look in φ_i
```

Figure 3: The procedure FINDCONSTANT.

and works as follows. It either succeeds in finding a subformula (which may be φ itself) that is a maximal constant subformula in any representation of the target C, and the value of the constant, or it reduces φ to a subformula with at most two-thirds of the original variables that evaluates $\mathbf{x}$ differently in φ and in any representation of C. The procedure FINDCONSTANT then continues recursively, using $O(\log n)$ queries on each level. This guarantees that after $O(\log^2 n)$ membership queries the procedure finds a subformula that is a maximal constant subformula in any representation of C, and the value of the constant. One can then find a substitution with a minimal number of variables that forces the given constant value of the subformula. This is a standard recursive computation that does not involve making queries.[5]

For a more detailed description, let us consider the version of FINDCONSTANT in Figure 3. At the bottom of the recursion no queries have to be asked: if $\mathbf{x}$ is a counterexample to a formula consisting of a single variable, then the revision must be fixing this variable to the constant different from $\mathbf{x}$.

If the input formula has more than one variable, then FINDCONSTANT starts by making sure that $\mathrm{MQ}(\mathbf{0}) = 0$ and $\mathrm{MQ}(\mathbf{1}) = 1$. Otherwise, the whole subformula is identically true or false.

Then FINDCONSTANT asks the membership queries $\mathrm{MQ}(\mathbf{0}, \alpha)$ and $\mathrm{MQ}(\mathbf{1}, \alpha)$, where α is the partial truth assignment sensitizing some approximately half-size subformula φ' of φ. Depending on the outcome of these queries, we distinguish two cases.

Case I: $\mathrm{MQ}(\mathbf{0}, \alpha) = \mathrm{MQ}(\mathbf{1}, \alpha) = c$ for $c = 0$ or 1. Then for every truth assignment $\mathbf{y}$ to the variables of φ' it holds that $\mathrm{MQ}(\mathbf{y}, \alpha) = c$, by monotonicity. Thus the subformula corresponding to φ' or to some subformula containing φ' computes the constant function c, in any representation of the target function.

Procedure GROWFORMULA, which we now informally describe, gets a read-once formula φ, a subformula φ', and a constant c as input such that $\mathrm{MQ}(\mathbf{0}, \alpha) = \mathrm{MQ}(\mathbf{1}, \alpha) = c$. Using $O(\log n)$ membership queries GROWFORMULA outputs a maximal subformula containing φ' such that the corresponding subformula is identical to the constant c in any representation of the target.

Let us assume *w.l.o.g.* that $c = 1$. Let α_i for $i = 0, \ldots, r$ be the partial truth assignment that is identical to α for $X_1, \ldots, X_i$, leaves the variables in Y unassigned, and assigns 0 to all the other variables. Then $(\mathbf{0}, \mathbf{0}) = (\mathbf{0}, \alpha_0) \leq (\mathbf{0}, \alpha_1) \leq (\mathbf{0}, \alpha_2) \leq \cdots \leq (\mathbf{0}, \alpha_r) = (\mathbf{0}, \alpha)$ in the usual componentwise ordering of $\{0,1\}^n$, and it holds that $\mathrm{MQ}(\mathbf{0}, \alpha_0) = 0$ and $\mathrm{MQ}(\mathbf{0}, \alpha_r) = 1$.

Using binary search, we can find an i ($1 \leq i \leq r$) such that $\mathrm{MQ}(\mathbf{0}, \alpha_{i-1}) = 0$ and $\mathrm{MQ}(\mathbf{0}, \alpha_i) = 1$. The only difference between the truth assignments $(\mathbf{0}, \alpha_{i-1})$ and $(\mathbf{0}, \alpha_i)$ is that the variables in X_i are off in $(\mathbf{0}, \alpha_{i-1})$ and they may be on in $(\mathbf{0}, \alpha_i)$. In fact, they must be on, as otherwise $(\mathbf{0}, \alpha_{i-1}) = (\mathbf{0}, \alpha_i)$. It also follows that $\circ_i$ is AND. Thus, on one hand, it must be the case that $\psi_i(\mathbf{0}) = 0$ and $\psi_i(\mathbf{1}) = 1$ in any representation of the target concept, where ψ_i is the subformula corresponding to φ_i in the target. On the other hand, it must be the case that the input to $\circ_i$ on the path P provided by its child on the path is equal to 1 in both cases. As the variables in this subformula are all set to 0, this subformula must compute the constant 1 function. The inputs $(\mathbf{0}, \alpha_{i-1})$ and $(\mathbf{0}, \alpha_i)$ demonstrate that no larger subformula computes a constant function.

Case II: otherwise, it must be the case that $\mathrm{MQ}(\mathbf{0}, \alpha) = 0$ and $\mathrm{MQ}(\mathbf{1}, \alpha) = 1$. Then for every truth assignment $\mathbf{y}$ to the variables of φ' it holds that

$$\mathrm{MQ}(\mathbf{y}, \alpha) = \psi'(\mathbf{y}), \tag{4}$$

[5] For example, let us assume that the final gate in the formula is an AND. Then the minimal number of variables needed to fix the output to 0 is obtained by taking the minimum of the minimal number of variables needed to fix the left and right subformulas to 0. The minimal number of variables needed to fix the output to 1 is the sum of the minimal number of variables needed to fix the left and right subformulas to 1. This recursion can of course also be used to find the substitution itself, not just the number of variables in it.

where ψ' is the subformula corresponding to φ' in any representation of the target concept. It is at this point that we start considering the counterexample $\mathbf{x}$, which we write as $(\mathbf{x}_1, \mathbf{x}_2)$, corresponding to its φ'-partition. Using Equation 4, we can compare the known value of $\varphi'(\mathbf{x}_1)$ to $\psi'(\mathbf{x}_1)$ by asking the membership query $\mathrm{MQ}(\mathbf{x}_1, \alpha)$. There are two possibilities.

Case II.1: $\mathrm{MQ}(\mathbf{x}_1, \alpha) = \psi'(\mathbf{x}_1) \neq \varphi'(\mathbf{x}_1)$. Then $\mathbf{x}_1$ is a counterexample to the hypothesis φ' for the target concept ψ'. Thus we can continue recursively, to find a constant substitution in a problem which has at most two-thirds of the original variables. Note that by Equation 4 we can use the *original* membership queries to simulate membership queries to the new target concept.

Case II.2: $\mathrm{MQ}(\mathbf{x}_1, \alpha) = \psi'(\mathbf{x}_1) = \varphi'(\mathbf{x}_1) = d$.

We use the procedure FINDFORMULA to find a subformula that can be processed recursively, analogously to φ' in the previous case. First we have to introduce some additional notation, based on writing φ as in Equation 3. Put $\mathbf{x}_2 = (\mathbf{x}_{2,1}, \ldots, \mathbf{x}_{2,r})$, where $\mathbf{x}_{2,i}$ corresponds to the variables in X_i. As in Case I, let ψ_i be the subformula corresponding to φ_i in some representation ψ of the target.

Let y_i (resp., z_i) be the value computed at $\circ_i$ in φ (resp., ψ) on input $\mathbf{x}$, for $i = 1, \ldots, r$, and let $y_{r+1} = z_{r+1} = d$. Then by definition it holds that

$$\begin{aligned} y_i &= y_{i+1} \circ_i \varphi_i(\mathbf{x}_{2,i}) \\ z_i &= z_{i+1} \circ_i \psi_i(\mathbf{x}_{2,i}) \end{aligned} \tag{5}$$

for $i = 1, \ldots, r$. Also, $y_1 = \varphi(\mathbf{x}) \neq \psi(\mathbf{x}) = z_1$.

Lemma 8 *Let β_i be the partial truth assignment that assigns $\mathbf{x}_{2,j}$ to X_j for $j = i, \ldots, r$ and is otherwise identical to α. Then $z_i = \mathrm{MQ}(\mathbf{x}_1, \beta_i)$.*

Proof. This follows from the properties of sensitizing assignments and the definition of z_i. □

Now we can describe the procedure FINDFORMULA. It was noted that $y_{r+1} = z_{r+1}$ and $y_1 \neq z_1$. FINDFORMULA performs a binary search to find an i $(1 \leq i \leq r)$ such that $y_{i+1} = z_{i+1}$ and $y_i \neq z_i$, and it returns i. The search is based on using Lemma 8 to evaluate z_i with membership queries. The values of y_i can be determined from φ without any queries. Equation 5 implies that for this i it holds that $\varphi_i(\mathbf{x}_{2,i}) \neq \psi_i(\mathbf{x}_{2,i})$. The procedure FINDCONSTANT then continues by a recursive call on φ_i using the counterexample $\mathbf{x}_{2,i}$.

We must show how to make membership queries in the recursive call. The following lemma, whose proof we omit, shows how to set the variables other than those in X_i so that the membership queries can be made.

Lemma 9 *Let γ_i be the partial truth assignment that assigns 1 (resp., 0) to Y and to all X_j with $j > i$ if $\circ_i$ is AND (resp., OR) and is identical to α on X_j for $1 \leq j < i$. For every truth assignment $\mathbf{w}$ to X_i, it holds that*

$$\mathrm{MQ}(\mathbf{w}, \gamma_i) = \psi_i(\mathbf{w}).$$

We claim that FINDCONSTANT finds a maximal constant subformula. This can be proved by induction on the number of variables, making use of the argument given for GROWFORMULA and of the fact that none of the subformulas rooted at any $\circ_i$ in φ can be constant.

We use $O(\log^2 n)$ membership queries in one top-level, nonrecursive call of the procedure, as the depth of the recursion is $O(\log n)$, and $O(\log n)$ membership queries are asked on each level. This completes the description and analysis of the procedure FINDCONSTANT.

The learning algorithm LEARNREADONCE is given in Figure 4.

LEARNREADONCE (φ)
  while $(\mathbf{x} = \mathrm{EQ}(\varphi)) \neq$ "correct"
    $\sigma = \mathrm{FINDCONSTANT}(\varphi, \mathbf{x})$
    $\varphi = \varphi\sigma$
  return φ

Figure 4: Algorithm LEARNREADONCE

In each call of FINDCONSTANT by LEARNREADONCE, a substitution is found and the corresponding maximal constant subformula is eliminated from the formula. Thus the updated formula contains fewer variables. As the membership queries always refer to truth assignments to the original set of variables, the new membership queries have to assign some values to the eliminated variables as well. The construction implies that these variables are irrelevant, and therefore their values can be arbitrary.

The correctness and the complexity bound of the algorithm follows by noting that in each call to FINDCONSTANT by LEARNREADONCE we find a new maximal constant subformula that is disjoint from the previous ones and thus it contains at least one new variable that is fixed to a constant.

Thus the number of calls LEARNREADONCE makes to FINDCONSTANT is at most the revision distance between φ and the target. The query bound then follows from the fact that FINDCONSTANT uses $O(\log^2 n)$ membership queries. □

5 A HARD MONOTONE DNF

Consider the variables $x_1, \ldots, x_n, y_1, \ldots, y_n$ and let $\varphi_n = t_1 \vee \cdots \vee t_n$, where, for $i = 1, \ldots, n$,

$$t_i = x_1 \wedge \cdots \wedge x_{i-1} \wedge x_{i+1} \wedge \cdots \wedge x_n \wedge y_i.$$

Theorem 10 *The formula φ_n requires at least $n - 1$ membership and equivalence queries to be revised, even if it is known that exactly one literal y_i is deleted, and every revised version of φ_n may be asked as a hypothesis.*

Proof. We describe an adversary strategy for answering the queries of any learning algorithm. Let ψ_i be the formula obtained from φ_n by deleting the single occurrence of y_i. Thus, initially the set of possible target concepts is $\Psi = \{\psi_1, \ldots, \psi_n\}$. Using the adversary strategy described below, each query eliminates at most one concept from Ψ, and this implies the claimed lower bound.

Consider a truth assignment $(\mathbf{a}, \mathbf{b})$, where $\mathbf{a}$ is a truth assignment to the x_i's and $\mathbf{b}$ is a truth assignment to the y_i's. Then the membership query $\mathrm{MQ}(\mathbf{a}, \mathbf{b})$ is answered as follows. If $\mathbf{a}$ has at most $n - 2$ components equal to 1, then

$\text{MQ}(\mathbf{a}, \mathbf{b}) = 0$. This does not eliminate any concepts from Ψ. If $\mathbf{a}$ has $n-1$ components equal to 1 and $a_i = 0$, then $\text{MQ}(\mathbf{a}, \mathbf{b}) = b_i$. If $b_i = 1$ then this does not eliminate any concept from Ψ. If $b_i = 0$ then ψ_i is eliminated from Ψ. If $\mathbf{a} = \mathbf{1}$ then $\text{MQ}(\mathbf{a}, \mathbf{b}) = 1$. This does not eliminate any concept from Ψ.

Now consider an equivalence query $EQ(\theta)$, where θ is a revised version of φ_n. Thus θ may be obtained from φ_n by deleting any number of literals or terms. If θ contains a term t with at most $n-2$ of the x_i's, then return the negative counterexample $(\mathbf{t}, \mathbf{1})$. This does not eliminate any concept from Ψ. If θ contains a term t such that t contains all the x variables except x_i and it does not contain y_i, then return the negative counterexample $(\mathbf{t}, \mathbf{f}_i)$, where $\mathbf{f}_i$ has all 1's except in the ith position. This eliminates only ψ_i from Ψ. Finally, if θ is φ_n itself, then return the positive counterexample $(\mathbf{1}, \mathbf{0})$. This does not eliminate any concept from Ψ. □

6 A MORE GENERAL ERROR MODEL

So far we considered only errors corresponding to the deletion of literals and terms. In practical theory revision algorithms one also has to deal with other types of errors such as the replacement of a variable with another one, or the addition of a variable or a term. Some of these error types are hard to define in general, and one has to be careful with their definition in particular cases (see, e.g. [5, 15]). Replacements and additions appear to be harder to handle than deletions.

Let the variables $x_1, \dots, x_n$ be given. We consider monotone conjunctions and *parity functions*. A parity function is the exclusive-or of a subset of the variables, or the complement of such a function. Thus a parity function can be written as $\mathbf{u} \cdot \mathbf{x} \oplus a$, where $\mathbf{u} \in \{0,1\}^n$, $a \in \{0,1\}$, and $\cdot$ denotes the mod 2 inner product of two vectors. Given a monotone conjunction or a parity function, we now allow the deletion of a variable, the replacement of a variable by a constant or *another variable*, the *addition* of a variable, and for parity, also the addition of the constant 1. Given a monotone conjunction or a parity function φ, we denote by $\mathcal{R}_\varphi$ the class of monotone conjunctions, resp., parity functions that can be obtained from φ using the enlarged set of revision operators, and we denote by $\mathcal{C}_\varphi$ the corresponding concept classes. Thus, clearly $\mathcal{C}_\varphi$ is the class of all monotone conjunctions, resp., parity functions over the variables $x_1, \dots, x_n$. In these cases, unlike the rest of the paper, the concept classes do not depend on the initial formula. The only role played by the revision operator is to determine the performance metric: if the target concept can be obtained with a few revisions then we have to identify it with few queries.

Proposition 11 *Every monotone conjunction φ has a revision algorithm that uses $O(d \log n)$ queries, where d is the revision distance between φ and the target concept.*

Proof. Asking membership queries to $\mathbf{0}$ and $\mathbf{1}$, it can be determined if the target concept is identically true or false. If neither occurs, we ask an equivalence query to φ. If a positive counterexample $\mathbf{x}$ is returned, than some variable occurring in φ is set to 0 in $\mathbf{x}$. This variable must be deleted. If a negative counterexample $\mathbf{x}$ is returned, then some variable that is set to 0 in $\mathbf{x}$ must be added to φ. Such a variable can be found with $O(\log n)$ membership queries as follows. Switching on the 0-bits of $\mathbf{x}$ one by one, we get an increasing sequence of vectors. Using binary search, we can find the point where the classification changes from 0 to 1 in this sequence, and the bit that is just switched on corresponds to such a variable. This process can be iterated until the target concept is found. □

Theorem 12 *Every parity function φ has a revision algorithm that uses $O(d \log n)$ queries, where d is the revision distance between φ and the target concept.*

Proof. Let $\varphi(\mathbf{x}) = \mathbf{u}_\varphi \cdot \mathbf{x} \oplus a$ be the parity function to be revised, and let $\psi(\mathbf{x}) = \mathbf{u}_\psi \cdot \mathbf{x} \oplus b$ be the target concept. As $\psi(\mathbf{0}) = b$, the value of b can be determined with the single equivalence query $\text{MQ}(\mathbf{0})$. If $a \neq b$ then we can change φ to its complement to achieve $a = b$, and if $a = b = 1$ then we can reverse labels. Thus it may be assumed that $a = b = 0$. The vectors $\mathbf{u}_\varphi$ and $\mathbf{u}_\psi$ differ in at most $2d$ bits.

The learning algorithm starts with the equivalence query φ. Let $\mathbf{x}$ be the counterexample received for this query. As $a = b = 0$, it holds that $\mathbf{x} \neq \mathbf{0}$. Our goal now is to find a counterexample containing exactly one 1. Let $\mathbf{x}_1$, resp. $\mathbf{x}_2$, be obtained from $\mathbf{x}$ by switching off the first, resp. second, half of the 1 components in $\mathbf{x}$. Notice that $\mathbf{x} = \mathbf{x}_1 \oplus \mathbf{x}_2$, and so

$$\begin{aligned} \varphi(\mathbf{x}_1) \oplus \varphi(\mathbf{x}_2) &= \varphi(\mathbf{x}_1 \oplus \mathbf{x}_2) = \varphi(\mathbf{x}) \\ &\neq \psi(\mathbf{x}) = \psi(\mathbf{x}_1 \oplus \mathbf{x}_2) \\ &= \psi(\mathbf{x}_1) \oplus \psi(\mathbf{x}_2), \end{aligned}$$

so exactly one of $\varphi(\mathbf{x}_1) \neq \psi(\mathbf{x}_1)$ and $\varphi(\mathbf{x}_2) \neq \psi(\mathbf{x}_2)$ hold. Thus exactly one of $\mathbf{x}_1$ and $\mathbf{x}_2$ is a counterexample, and one membership query will tell us which one is the counterexample. Continuing this process, a counterexample with a single 1 component can be found with $O(\log n)$ membership queries. The variable corresponding to the 1 component must be one of the variables where φ and ψ differ. Hence ψ can be found by repeating this procedure $O(d)$ times. □

7 CONCLUDING REMARKS

The revision algorithms presented in Sections 3 and 4 suggest the following general situation: if a monotone DNF has either few terms, or only small terms, or disjoint terms, then it can be revised efficiently. We think that we can extend the 2-term monotone DNF revision algorithm to k-term monotone DNF, which would be in accord with this picture. On the other hand, the lower bound of Section 5 shows that a monotone DNF expression with many large terms having large overlap may be hard to revise. In general, it would be interesting to understand the relationship between the revision complexity of a monotone DNF and its combinatorial properties. The revision complexity of general DNF is open even for 2-term DNF. Extending the positive results to addition type revision operators would also be interesting. A next step here is to consider monotone read-once DNF expressions, where we have some partial results.

For practical applications, it would be very interesting to find efficient revision algorithms for Horn-sentences, that is, conjunctions of clauses with at most one unnegated literal [3, 5, 23] even in restricted cases such as having small clauses or small overlap. Another direction for further work, also relevant for practice, is theory revision in predicate logic. Some initial positive results are obtained in [5].

ACKNOWLEDGMENT

We thank Moshe Koppel. This paper would never have come into being without initial discussions with him on the topic of theory revision.

We also thank the referees for their helpful remarks. In particular, we thank them for suggesting we include Proposition 1, and for proposing a simpler algorithm with a better bound for Theorem 12.

References

[1] Z. Alexin, T. Gyimóthy, and H. Boström. IMPUT: An interactive learning tool based on program specialization. *Intelligent Data Analysis*, 1(4), 1997. ⟨http://www-east.elsevier.com/ida/⟩.

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

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

[4] D. Angluin, L. Hellerstein, and M. Karpinski. Learning read-once formulas with queries. *J. ACM*, 40:185–210, 1993.

[5] S. Argamon-Engelson and M. Koppel. Tractability of theory patching. *Journal of Artificial Intelligence Research*, 8:39–65, 1998.

[6] A. Blum, L. Hellerstein, and N. Littlestone. Learning in the presence of finitely or infinitely many irrelevant attributes. *J. of Comput. Syst. Sci.*, 50(1):32–40, 1995. Earlier version in 4th COLT, 1991.

[7] N. Bshouty and L. Hellerstein. Attribute-efficient learning in query and mistake-bound models. *J. Comput. Syst. Sci.*, 56:310–319, 1998.

[8] J. Case, S. Kaufmann, E. Kinber, and M. Kummer. Learning recursive functions from approximations. *J. Comput. Syst. Sci.*, 55:183–196, 1997.

[9] R. Davis and W. Hamscher. Model-based reasoning: Troubleshooting. In H. E. Shrobe and the American Association for Artificial Intelligence, editors, *Exploring Artificial Intelligence: Survey Talks from the National Conferences on Artificial Intelligence*, chapter 8, pages 297–346. Morgan Kaufmann, San Mateo, CA, 1988.

[10] T. Eiter and G. Gottlob. On the complexity of propositional knowledge base revision, updates, and counterfactuals. *Artificial Intelligence*, 57:227–270, 1992.

[11] P. Gärdenfors. *Knowledge in Flux*. Bradford Books/MIT Press, Detroit/Cambridge, Mass., 1988.

[12] D. P. Helmbold and P. M. Long. Tracking drifting concepts by minimizing disagreements. *Machine Learning*, 14(1):27–45, 1994.

[13] S. Jain and A. Sharma. Learning in the presence of partial explanations. *Inform. Comput.*, 95-2:162–191, 1991.

[14] Z. Kohavi. *Switching and Finite Automata Theory*. McGraw-Hill, New York, NY, second edition, 1978.

[15] M. Koppel, R. Feldman, and A. M. Segre. Bias-driven revision of logical domain theories. *Journal of Artificial Intelligence Research*, 1:159–208, 1994.

[16] J. E. Laird, A. Newell, and P. S. Rosenbloom. SOAR: An architecture for general intelligence. *Artificial Intelligence*, 33(1):1–64, Sept. 1987.

[17] W. Maass and G. Turán. Lower bound methods and separation results for on-line learning models. *Machine Learning*, 9:107–145, 1992.

[18] R. A. Marcotte, M. J. Neiberg, R. L. Piazza, and L. J. Holtzblatt. Model-based diagnostic reasoning using VHDL. In J. M. Schoen, editor, *Performance and Fault Modeling with VHDL*, chapter 6, pages 304–399. Prentice Hall, Englewood Cliffs, NJ, 1992.

[19] J. de Kleer, A. K. Mackworth, and R. Reiter. Characterizing diagnoses and systems. *Artificial Intelligence*, 56:197–222, 1992.

[20] R. J. Mooney. A preliminary PAC analysis of theory revision. In *Computational Learning Theory and Natural Learning Systems, Volume III: Selecting Good Models*, chapter 3, pages 43–53. MIT Press, 1995.

[21] D. Ourston and R. J. Mooney. Theory refinement combining analytical and empirical methods. *Artificial Intelligence*, 66:273–309, 1994.

[22] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32:57–95, 1987.

[23] B. L. Richards and R. J. Mooney. Automated refinement of first-order Horn-clause domain theories. *Machine Learning*, 19:95–131, 1995.

[24] R. L. Rivest and R. Sloan. A formal model of hierarchical concept learning. *Inform. Comput.*, 114:88–114, 1994.

[25] E. Y. Shapiro. *Algorithmic Program Debugging*. MIT Press, Cambridge, MA, 1983.

[26] G. G. Towell and J. W. Shavlik. Extracting refined rules from knowledge-based neural networks. *Machine Learning*, 13:71–101, 1993.

[27] I. Wegener. *The Complexity of Boolean Functions*. Wiley–Teubner, 1987.

[28] S. Wrobel. Concept formation during interactive theory revision. *Machine Learning*, 14(2):169–191, 1994.

[29] S. Wrobel. First order theory refinement. In L. De Raedt, editor, *Advances in ILP*, pages 14–33. IOS Press, Amsterdam, 1995.