Journal of Computer and System Sciences, 52(2):201-213, April, 1996.

Learning Sparse Multivariate Polynomials over a Field with Queries and Counterexamples

Robert E. Schapire
AT&T Bell Laboratories
Murray Hill, NJ 07974
schapire@research.att.com

Linda M. Sellie*
University of Chicago
Chicago, IL 60637
sellie@research.att.com

Abstract. We consider the problem of learning a polynomial over an arbitrary field F defined on a set of boolean variables. We present the first provably effective algorithm for exactly identifying such polynomials using membership and equivalence queries. Our algorithm runs in time polynomial in n, the number of variables, and t, the number of nonzero terms appearing in the polynomial. The algorithm makes at most $nt+2$ equivalence queries, and at most $(nt+1)(t^2+3t)/2$ membership queries. Our algorithm is equally effective for learning a generalized type of polynomial defined on certain kinds of semilattices. We also present an extension of our algorithm for learning multilinear polynomials when the domain of each variable is the entire field F.

1 Introduction

We consider the problem of learning a polynomial f over an arbitrary field F defined on a set of boolean variables. Thus, we are interested in the learnability of multivariate polynomials over a field F when the domain of each variable x_i has been restricted to the values 0 and 1 (the additive and multiplicative identity elements of F). This problem models a learning situation in which examples are most naturally described by vectors of boolean attributes, but in which the behavior of the function to be learned is most easily described by a polynomial in the boolean values.

We consider the learnability of such polynomials in a model introduced by Angluin [1] in which the learning algorithm has two forms of access to the unknown target polynomial: The first of these is a so-called *membership oracle* which evaluates the target polynomial f on any variable-setting $\mathbf{a}$ of the learner's choosing and returns the result $f(\mathbf{a})$. The second form of access is a so-called *equivalence oracle* which accepts as input a hypothesis polynomial h conjectured by the learner to be equivalent to the target polynomial. If h and f are functionally equivalent, then the equivalence oracle replies "EQUIVALENT" (in which case we say that the target has been *exactly identified*); otherwise, the equivalence oracle provides a *counterexample*,

*Currently visiting AT&T Bell Laboratories.

a setting $\mathbf{a}$ of the variables on which the hypothesis and target polynomials evaluate to different values, i.e., for which $h(\mathbf{a}) \neq f(\mathbf{a})$.

The main result of this paper is a provably correct algorithm for exactly identifying polynomials on a set of boolean variables over an arbitrary field F using equivalence and membership queries. Our algorithm runs in time polynomial in n, the total number of variables, and t, the number of nonzero coefficients of terms in f (called the *sparcity* of f). The algorithm makes no more than $nt + 2$ equivalence queries and roughly $nt^3/2$ membership queries. This is the first provably efficient algorithm for this problem.

As an added feature, our algorithm uses only equivalence queries that are "proper" in the sense that each conjectured hypothesis is itself a t-sparse polynomial.

By applying a technique of Angluin [1], our result can be extended to Valiant's so-called probably approximately correct (PAC) learning model [20]. Specifically, Angluin shows that an equivalence oracle can always be replaced by a source of random examples, in which case the learning algorithm will successfully find a hypothesis that is a good approximation of the target polynomial with respect to the distribution on the random examples. Thus, our result implies that polynomials on the boolean domain over any field F can be learned in the PAC model with membership queries.

Of particular interest is the case in which F is taken to be GF(2), the finite field of integers modulo two. Such polynomials can be viewed equivalently as depth-two boolean formulas consisting of the XOR of several monotone monomials (i.e., conjunctions of unnegated variables).

The class of polynomials on the boolean domain includes a wide variety of functions. For instance, we show that logarithmic-depth decision trees can be computed by such polynomials.

One natural extension of our problem is to allow the domain of each variable to be the entire field F, rather than the restricted domain $\{0, 1\}$. We show that our algorithm can be used as a subroutine to learn a polynomial $f : F^n \to F$, provided that f is multilinear (so that each variable has degree at most 1). It is an open problem whether non-multilinear polynomials are generally learnable in this model, for an arbitrary field F.

The proof of the correctness of our algorithm relies on certain lattice-theoretic properties of the boolean domain $\{0, 1\}^n$, suggesting a natural extension of the domain of the target function. Specifically, we prove that our algorithm can learn a function $f : X \to F$ where F is a field and X is a meet-semilattice with no infinite chains, and where f is represented by a generalized type of "polynomial" on X.

Previous work. The problem of learning or inferring a polynomial from examples has an extensive history. For certain "large" fields, such as $\mathbb{R}$, efficient algorithms are known for identifying a polynomial from sample points of the learner's choosing. However, these algorithms typically require that it be possible to assign each variable many different values (hence the need for the field to be "large"). Thus, such algorithms are essentially useless if the domain of the function is the boolean domain since in this case each variable can only be assigned one of two values. Indeed, for small fields such as GF(2), most of the previous work has demonstrated that polynomials are hard to learn in various models, except in special cases.

In the PAC model (in the absence of membership and equivalence queries), Blum and Singh [5] and Fischer and Simon [11] show that it is computationally hard to learn t-sparse polynomials over GF(2) for any fixed $t \geq 2$ (assuming RP $\neq$ NP, and also assuming that the hypothesis must be expressed as a t-sparse polynomial). Their result also implies that it is hard to learn t-sparse GF(2)-polynomials using a "proper" equivalence oracle. Their proofs can be extended to show that t-sparse polynomials over any field F cannot be learned in either of these learning models, for any fixed $t \geq 2$, even if the domain is limited to $\{0,1\}^n$.

On the other hand, Blum and Singh show that t-sparse GF(2)-polynomials can be efficiently approximated in the PAC model using a DNF-representation in time poly(n^t). Also, Fischer and Simon give an efficient algorithm for the special case in which each term of the target polynomial has size at most k, where k is a constant. (Although proved for GF(2), this algorithm generalizes easily to any field F.) Their algorithm uses linear-algebraic techniques to learn such polynomials in time poly(n^k), either in the PAC model, or using equivalence queries only.

As mentioned above, much previous work has also focused on the problem of "interpolating" a polynomial, i.e., on the problem of exactly identifying a polynomial using membership queries only. As noted above, most of this work has dealt with large fields, such as the real numbers. For instance, Zippel [22, 23], Ben-Or and Tiwari [3], and Mansour [18] give efficient algorithms for interpolating sparse multivariate polynomials over such fields.

Grigoriev, Karpinski and Singers [13] and Clausen et al. [8] consider the problem of interpolating a sparse polynomial over various finite fields (see also the related work of Dress and Grabmeier [9] and Dür and Grabmeier [10]). However, for small fields (such as GF(p), where p is a small prime), their algorithms are efficient only if queries can be made over a larger extension field. For the field GF(2), when no such extension is made, Clausen et al. show that t-sparse, n-variable polynomials can be efficiently interpolated using poly($n^{\log t}$) queries, and that the number of queries required is essentially optimal. These bounds are also proved by Roth and Benedek [19], and are refined by Hellerstein and Warmuth [15] who show that poly($n^{\log k}$) queries suffice where k is the maximum number of terms in which any variable appears.

The lower bound results mentioned above can easily be extended to show that interpolation of a t-sparse polynomial over any field F requires at least $n^{\Omega(\log t)}$ boolean membership queries. Thus, for our problem, in which the goal is inference of an F-valued function on the boolean domain, membership queries alone cannot suffice for polynomial-time identification.

Thus, in sum, the previous research has demonstrated that it is hard or impossible to learn a sparse polynomial over any field on a set of boolean variables in polynomial time using membership queries only, random examples only, or proper equivalence queries only. In contrast, our result demonstrates the tractability of the learning problem if both membership queries and equivalence queries (or random examples) are available to the learner.

In Section 2, we describe our representation of polynomials, and their generalization to arbitrary semilattices. Our learning algorithm is described in Section 3. Although the algorithm is appealingly simple, its analysis is quite involved and relies

on certain combinatorial facts proved in Section 4. We apply these combinatorial results to give a full analysis of the algorithm in Section 5, and present extensions and applications of the main algorithm in Section 6.

2 Polynomials and Semilattices

Let f be a multivariate polynomial over a field F defined on n boolean variables. Then f is a function mapping the boolean domain $\{0,1\}^n$ into the field F, where 0 and 1 are the additive and multiplicative identity elements of F. The *sparcity* of f is the number of nonzero coefficients appearing in f; if f has sparcity at most t, then f is said to be *t-sparse*.

Since $x^2 = x$ for $x \in \{0,1\}$, we can assume without loss of generality that f is multilinear, i.e., that no variable has exponent greater than one in f. It then becomes natural to associate with each monomial the characteristic vector of indices of variables that appear in that monomial. For instance, if $n = 5$ then the monomial $x_1x_3x_4$ is associated with the vector 10110. Conversely, every vector $\mathbf{a} \in \{0,1\}^n$ is associated with a monomial which we denote by $\mathbf{x}^{\mathbf{a}}$ (for example, the monomial x_2x_4 could also be written $\mathbf{x}^{01010}$).

Using this notation, every t-sparse polynomial can be written in the form

$$f(\mathbf{x}) = \sum_{i=1}^{t} c_i \mathbf{x}^{\mathbf{t}_i} \tag{1}$$

for some $c_i \in F$ and $\mathbf{t}_i \in \{0,1\}^n$.

Our algorithm depends crucially on the lattice properties of the boolean domain $\{0,1\}^n$. To make this dependence explicit, we will prove our results using an extended notion of "polynomial" for more general domains. Specifically, we will show that such polynomials can be learned if the domain is a meet-semilattice with no infinite chains (defined below).

Let X be a set partially ordered by $\leq$. We say that an element $\mathbf{c} \in X$ is the *meet* of $\mathbf{a}$ and $\mathbf{b}$ if for all $\mathbf{d} \in X$, $\mathbf{d} \leq \mathbf{c}$ if and only if $\mathbf{d} \leq \mathbf{a}$ and $\mathbf{d} \leq \mathbf{b}$. Thus, the meet of $\mathbf{a}$ and $\mathbf{b}$, written $\mathbf{a} \wedge \mathbf{b}$, can be viewed as the greatest lower bound of $\mathbf{a}$ and $\mathbf{b}$. If the meet exists for every pair of elements $\mathbf{a}$ and $\mathbf{b}$ in X, then X is said to be a *meet-semilattice* (henceforth called simply a semilattice).

On the boolean domain $\{0,1\}^n$, it is natural to define a partial order in which $\mathbf{a} \leq \mathbf{b}$ if and only if $a_i \leq b_i$ for $i = 1, \ldots, n$ (and where we define the usual ordering on $\{0,1\}$ with $0 < 1$). With respect to this ordering, $\{0,1\}^n$ is clearly a semilattice. The meet $\mathbf{a} \wedge \mathbf{b}$ is the vector $\mathbf{c}$ where $c_i = a_i b_i$ for each i. (For instance, $01101 \wedge 10101 = 00101$.) Note also that the monomial $\mathbf{x}^{\mathbf{a}}$ is satisfied by $\mathbf{b}$ (i.e., is equal to 1) if and only if $\mathbf{a} \leq \mathbf{b}$.

For a semilattice X, and for any point $\mathbf{a} \in X$, we define a function $\pi_{\mathbf{a}} : X \to \{0,1\}$ by the rule

$$\pi_{\mathbf{a}}(\mathbf{b}) = \begin{cases} 1 & \text{if } \mathbf{a} \leq \mathbf{b} \\ 0 & \text{otherwise.} \end{cases}$$

Analogous to the case where X is the boolean domain, we define a *polynomial over F on semilattice X* to be a function $f : X \to F$ of the form

$$f(\mathbf{x}) = \sum_{i=1}^{t} c_i \pi_{\mathbf{t}_i}(\mathbf{x}) \tag{2}$$

where $c_i \in F$ and $\mathbf{t}_i \in X$. As noted above, for the boolean domain, $\mathbf{x}^{\mathbf{a}} = \pi_{\mathbf{a}}(\mathbf{x})$ for any $\mathbf{a} \in \{0,1\}^n$; thus, the definitions given in equations (1) and (2) are equivalent for this domain. Also, consistent with our earlier notion of sparcity, we say that a polynomial of the form given in equation (2) is *t-sparse*, and, if each c_i is nonzero, that the polynomial has *sparcity t*.

For a function f, we will use the notation $\tilde{f}(\mathbf{a})$ to denote the coefficient (possibly zero) of $\pi_{\mathbf{a}}(\mathbf{x})$. Thus,

$$f(\mathbf{x}) = \sum_{\mathbf{a} \in X} \tilde{f}(\mathbf{a}) \pi_{\mathbf{a}}(\mathbf{x}). \tag{3}$$

(If f is a polynomial, then it has a finite number of nonzero coefficients, so this sum will be well defined even if X is infinite.) Equivalently, by the definition of $\pi_{\mathbf{a}}$,

$$f(\mathbf{x}) = \sum_{\substack{\mathbf{a} \in X \\ \mathbf{a} \leq \mathbf{x}}} \tilde{f}(\mathbf{a}).$$

As an aside, we note that if X is finite, then every function $f : X \to F$ can be written uniquely in the form of equation (3). This is because the collection of functions $\{\pi_{\mathbf{a}} : \mathbf{a} \in X\}$ forms a linearly independent basis for the $|X|$-dimensional space of all functions mapping X into F.

If X has any minimal element, then this element must be unique, and it must be smaller than every other element of X. Such an element, if it exists, is called a *bottom element*, and it is denoted $\perp$.

Two elements $\mathbf{a}, \mathbf{b} \in X$ are *comparable* if $\mathbf{a} \leq \mathbf{b}$ or $\mathbf{b} \leq \mathbf{a}$. A subset $C \subseteq X$ is a *chain* if every pair of elements in C are comparable. The *length* of such a chain is $|C| - 1$. Thus, a length-n chain can be viewed as a sequence $\mathbf{a}_0 < \mathbf{a}_1 < \cdots < \mathbf{a}_n$.

If $\perp$ exists, then the *height* of a point $\mathbf{a} \in X$, denoted $||\mathbf{a}||$, is defined to be the length of the longest chain starting at $\perp$ and ending at $\mathbf{a}$. The *height* of X is then the maximum height of any element in X, or equivalently, is the length of the longest chain in X. Thus, semilattice X has finite height if it contains no infinite chains.

On the boolean domain $\{0,1\}^n$, the bottom element is $\mathbf{0}$, the vector whose every component is 0. The height of any vector $\mathbf{a}$ is equal to the number of 1's in the vector $\mathbf{a}$, and the height of the entire domain is n.

3 The Algorithm

This section describes our algorithm for learning polynomials with queries. In later sections, we will show that this algorithm can infer any polynomial over a field F

on a finite-height semilattice X in time polynomial in the sparcity t of the target polynomial f, and in the height n of the semilattice X.

We assume throughout that there exist efficient procedures for computing $\mathbf{a} \wedge \mathbf{b}$, $||\mathbf{a}||$, and for deciding if $\mathbf{a} \leq \mathbf{b}$ for any $\mathbf{a}, \mathbf{b} \in X$. Such procedures clearly exist if $X = \{0,1\}^n$.

We begin by observing that it is easy to find a hypothesis that is consistent with a given set of labeled examples. Specifically, let $S \subseteq X$ be a finite set of examples labeled by f. We define a new polynomial h in terms of its coefficients as follows: For $\mathbf{a} \in S$, we define $\tilde{h}(\mathbf{a})$, the coefficient in h of $\pi_{\mathbf{a}}(\mathbf{x})$, using the inductive rule:

$$\tilde{h}(\mathbf{a}) = f(\mathbf{a}) - \sum_{\substack{\mathbf{a}' \in S \\ \mathbf{a}' < \mathbf{a}}} \tilde{h}(\mathbf{a}'). \tag{4}$$

If $\mathbf{a} \notin S$, then $\tilde{h}(\mathbf{a})$ is defined to be 0. Algorithmically, it is clear that all of the coefficients $\tilde{h}(\mathbf{a})$ can be computed by visiting all of the elements in S from the smallest to the largest; that is, we visit the elements in such an order that no element $\mathbf{a}$ is visited until every element smaller than $\mathbf{a}$ has been visited.

The resulting polynomial

$$h(\mathbf{x}) = \sum_{\mathbf{a} \in X} \tilde{h}(\mathbf{a}) \pi_{\mathbf{a}}(\mathbf{x}) = \sum_{\substack{\mathbf{a} \in X \\ \mathbf{a} \leq \mathbf{x}}} \tilde{h}(\mathbf{a}) \tag{5}$$

is called the *manifest hypothesis of S with respect to f*, and it is denoted $hyp_f(S)$. When f is clear from context we write simply $hyp(S)$.

As mentioned above, $hyp(S)$ is consistent with f on S (although nothing is guaranteed about its behavior on $X - S$):

Lemma 1 *Let $f : X \to F$, where X is a semilattice and F is a field. Let S be a finite subset of X. Then the polynomial $h = hyp_f(S)$ is consistent with f on S; that is, $f(\mathbf{a}) = h(\mathbf{a})$ for all $\mathbf{a} \in S$.*

Proof: Note that, by construction of h, for every $\mathbf{a} \in S$,

$$f(\mathbf{a}) = \sum_{\substack{\mathbf{a}' \in S \\ \mathbf{a}' \leq \mathbf{a}}} \tilde{h}(\mathbf{a}') = \sum_{\substack{\mathbf{a}' \in X \\ \mathbf{a}' \leq \mathbf{a}}} \tilde{h}(\mathbf{a}')$$

since $\tilde{h}(\mathbf{a}) = 0$ for $\mathbf{a} \notin S$. On the other hand, by equation (5), the right hand side of this equation is exactly equal to $h(\mathbf{a})$. ■

If $h = hyp_f(S)$, then let

$$terms_f(S) = \{\mathbf{a} \in X : \tilde{h}(\mathbf{a}) \neq 0\}.$$

That is, $terms_f(S)$ is the set of elements associated with the nonzero coefficients of $hyp(S)$. Clearly, $terms(S) \subseteq S$, and $|terms(S)|$ is exactly the sparcity of $hyp(S)$. (As with $hyp(S)$, we drop the subscript of $terms_f(S)$ when f is clear from context.)

```
LearnPoly
Given: access to oracles Equiv and Member for target polynomial f
Output: a hypothesis equivalent to f
 1  S ← ∅.
 2  loop
 3     c ← Equiv(hyp(S))
 4     if c = "equivalent" then halt and output hyp(S)
 5     repeat
 6        AddElement(c)
 7        c ← EasyCounterexample()
 8     until (c = "stable")
 9  end
```

Figure 1: An algorithm for learning polynomials over F.

Although the hypothesis $hyp(S)$ is consistent with a given sample S, its sparcity may be quite large relative to the target function f. For example, suppose that f is the constant function 1, $X = \{0,1\}^n$ and $S = \{\mathbf{a} \in X : ||\mathbf{a}|| = n/2\}$, i.e., S is the set of all examples with exactly $n/2$ 1's. Then $terms(S) = S$, so the hypothesis $hyp(S)$ has sparcity $\binom{n}{n/2} = 2^{\Omega(n)}$, even though the target function f has only a single nonzero term.

Because $hyp(S)$ may be so much larger than the target function, by Occam's Razor, we would not intuitively expect $hyp(S)$ to generalize well as a hypothesis for classifying elements outside of S. We show below how to find a much smaller consistent hypothesis. Specifically, we show how to construct hypotheses which are no larger than the target function, a technique which will help us in bounding the number of queries made by our algorithm.

The main idea of this technique for simplifying the hypothesis is to add examples to the set S, and to then recompute the manifest hypothesis. Note that adding an example $\mathbf{c}$ to the set S will change the manifest hypothesis if and only if $\mathbf{c}$ is a counterexample (that is, $hyp(S) \neq hyp(S \cup \{\mathbf{c}\})$ if and only if $f(\mathbf{c}) \neq hyp(S)(\mathbf{c})$).

Let t be the sparcity of f. A key fact, which we prove in a later section, states that if $|terms(S)| > t$ then there must exist a pair $\mathbf{a}, \mathbf{b} \in terms(S)$ for which $\mathbf{a} \wedge \mathbf{b}$ is a counterexample for $hyp(S)$. Thus, if $|terms(S)| > t$ then we can find a counterexample to $hyp(S)$ by asking membership queries for the elements $\mathbf{a} \wedge \mathbf{b}$ where $\mathbf{a}, \mathbf{b} \in terms(S)$. We call such a counterexample an *easily observable counterexample* since it can be easily detected using membership queries.

If $hyp(S)$ has no easily observable counterexamples, then we say that S is *stable* with respect to the target function f. Thus, S is stable if for every pair $\mathbf{a}, \mathbf{b} \in terms(S)$, $f(\mathbf{a} \wedge \mathbf{b}) = h(\mathbf{a} \wedge \mathbf{b})$ where $h = hyp(S)$. Stability is a key notion in the development that follows.

By repeatedly adding easily observable counterexamples to S, we can eventually stabilize S (since there are only a finite number of elements that can ever be added). Unfortunately, if we add the counterexamples in an arbitrary order, we could poten-

EASYCOUNTEREXAMPLE
Return: an easily observable counterexample, or the flag "STABLE" if none exists

```
 1  T ← ∅
 2  while terms(S) ≠ T
 3     a ← any minimum height element in terms(S) − T
 4     for b ∈ T
 5        query and record MEMBER(a ∧ b)
 6        if a ∧ b is a counterexample for hyp(S) then
 7           return a ∧ b
 8        else
 9           S ← S ∪ {a ∧ b}
10     end
11     T ← T ∪ {a}
12  end
13  return "STABLE"
```

Figure 2: The subroutine EASYCOUNTEREXAMPLE.

tially add an exponential number of new elements before stabilizing. However, we show later in the paper that if we add the counterexamples in a somewhat greedy fashion, with bias toward the choice of small counterexamples, then we are guaranteed to stabilize after adding at most a polynomial number of elements to S. (Intuitively, we want to choose small counterexamples since these are likely to be "closer" to a true term of the target formula.)

A high-level description of our algorithm is given in Figure 1. In the figure, EQUIV is an equivalence oracle which takes as input a conjectured hypothesis polynomial, and returns either the flag "EQUIVALENT" if the hypothesis is equal to the target, or a counterexample for the hypothesis (i.e., an element on which the hypothesis and target functions disagree). We will also be using a membership oracle MEMBER which takes as input an element $\mathbf{a} \in X$ and returns the value of the target on $\mathbf{a}$.

Our algorithm maintains a set of examples S which is stable prior to each equivalence query. After each counterexample $\mathbf{c}$ is received from the equivalence oracle, we use $\mathbf{c}$ to modify S (line 6), and then restabilize S by repeatedly finding easily observable counterexamples (line 7) and using them to again modify S.

The subroutine for finding easily observable counterexamples is called EASY-COUNTEREXAMPLE and is described in Figure 2. The idea of the subroutine is simple: we test pairs $\mathbf{a}, \mathbf{b} \in terms(S)$ until we find one for which $\mathbf{a} \wedge \mathbf{b}$ is a counterexample. The subroutine tests pairs in a straightforward greedy fashion so as to find the counterexample $\mathbf{a} \wedge \mathbf{b}$ for which $\max(||\mathbf{a}||, ||\mathbf{b}||)$ is minimized. (We will see later why it is important to add the non-counterexamples to S at line 9.) If no counterexample is found, then S must be stable, and the subroutine returns a flag indicating this fact.

Finally, we describe ADDELEMENT, the subroutine that modifies S using a counterexample $\mathbf{c}$; this subroutine is shown in Figure 3. Before adding $\mathbf{c}$ to S, the subrou-

ADDELEMENT
Input: a counterexample $\mathbf{c}$ for $hyp(S)$

```
1  ℓ ← ||c||.
2  repeat
3     for a ∈ terms(S) : ||a|| = ℓ
4        query and record MEMBER(a ∧ c)
5        if a ∧ c is a counterexample for hyp(S) then
6           c ← a ∧ c
7           exit "for" loop
8     end
9     ℓ ← min(ℓ − 1, ||c||).
10 until ℓ = 0
11 S ← S ∪ {c} ∪ {a ∧ c : a ∈ terms(S), ||a|| ≤ ||c||}.
```

Figure 3: The subroutine ADDELEMENT.

tine first determines if there is another element $\mathbf{a} \in terms(S)$ for which $\mathbf{a} \wedge \mathbf{c}$ is also a counterexample. If such an $\mathbf{a}$ can be found, then we replace $\mathbf{c}$ with the smaller counterexample $\mathbf{a} \wedge \mathbf{c}$. We will see later why this greedy approach is helpful for analyzing the performance of the algorithm.

Note that by adding a new counterexample to S, we may radically change the structure and size (i.e., sparcity) of $hyp(S)$. Thus, it is not immediately clear that this technique will allow us to make substantial progress towards stabilization of S. The proof that S can be stabilized quickly is given in the following sections and is based on certain combinatorial properties of stable sets.

4 Properties of Stable Sets

In this section, we prove various properties of stable sets that will enable us to prove bounds on the number of queries made by our algorithm. In addition to the notion of stability described in the last section, we will also be interested in a slightly stronger notion: We say that a finite set $S \subseteq X$ is *properly stable* with respect to a function f if $\mathbf{a} \wedge \mathbf{b} \in S$ for all $\mathbf{a}, \mathbf{b} \in terms_f(S)$. Lemma 1 implies that S is stable if it is properly stable.

We begin with a proof that the sparcity of the manifest hypothesis of a stable set cannot exceed the sparcity t of the target polynomial. This fact, which may be of independent combinatorial interest, will be used repeatedly in the analysis of our algorithm. For instance, because S is stable prior to each equivalence query, this implies that the equivalence queries used by our algorithm are "proper" in the sense that the conjectured hypotheses always belong to the target class of t-sparse polynomials.

Theorem 2 *Let f be a t-sparse polynomial over a field F on a semilattice X. Let S be a finite subset of X that is stable with respect to f. Then $|terms_f(S)| \leq t$.*

We present two very different proofs of this theorem, one below and the other in the appendix.

Before presenting the first proof, we state two algebraic lemmas. Although these lemmas are standard, we include brief proofs for completeness.

We say that an element $s \in F$ is a *square* in F if there exists an element $r \in F$ for which $r^2 = s$.

Lemma 3 *Let F be a field, and let $A \subseteq F$ be a finite set of elements of F. Then there exists a field E of which F is a subfield, and in which each element of A is a square.*

Proof: It suffices to prove the lemma in the case that A is a singleton since the general result then follows by induction on $|A|$. If A is the singleton $\{s\}$ (where, without loss of generality, s is not already a square in F) then the result follows, for instance, from Theorem 5.3.1 of Herstein [16] in which E is taken to be the field $F[x]/(x^2 - s)$. The result can also be proved more directly by formally creating a new element r not already in F, and by defining a new field $E = \{a + br : a, b \in F\}$ in which $r^2 = s$ by definition (similar to the manner in which the imaginary number $i = \sqrt{-1}$ is adjoined to $\mathbb{R}$ to obtain the complex field $\mathbb{C}$). It can be verified that E is indeed a field that satisfies the required properties. ■

We define addition between two vectors $\mathbf{a}, \mathbf{b} \in F^t$ in the usual way. Thus, $\mathbf{a} + \mathbf{b}$ is that vector $\mathbf{c} \in F^t$ for which $c_i = a_i + b_i$ for $i = 1, \ldots, t$. Similarly, $\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{t} a_i b_i$ is the standard inner product of $\mathbf{a}$ and $\mathbf{b}$.

Lemma 4 *Let $A \subseteq F^t$ be a set of vectors over a field F such that, for all $\mathbf{a}, \mathbf{b} \in A$:*

1. *$\mathbf{a} \cdot \mathbf{a} \neq 0$, and*
2. *if $\mathbf{a} \neq \mathbf{b}$ then $\mathbf{a} \cdot \mathbf{b} = 0$.*

Then $|A| \leq t$.

Proof: We claim that the elements of A are linearly independent. For if $\sum_{\mathbf{a} \in A} \lambda_{\mathbf{a}} \mathbf{a} = \mathbf{0}$ for some coefficients $\lambda_{\mathbf{a}} \in F$, then for any $\mathbf{b} \in A$,

$$0 = \mathbf{b} \cdot \mathbf{0} = \mathbf{b} \cdot \left(\sum_{\mathbf{a} \in A} \lambda_{\mathbf{a}} \mathbf{a} \right) = \sum_{\mathbf{a} \in A} \lambda_{\mathbf{a}} (\mathbf{b} \cdot \mathbf{a}) = \lambda_{\mathbf{b}} (\mathbf{b} \cdot \mathbf{b})$$

by property 2. This implies $\lambda_{\mathbf{b}} = 0$ by property 1.

Since A is a linearly independent subset of a t-dimensional vector space, it follows immediately that $|A| \leq t$. ■

Proof of Theorem 2: Let f be the polynomial

$$f(\mathbf{x}) = \sum_{i=1}^{t} c_i \pi_{\mathbf{t}_i}(\mathbf{x})$$

where each $c_i \in F$ and $\mathbf{t}_i \in X$. Let $h = hyp_f(S)$.

We assume that S is properly stable. We make this assumption without loss of generality since adding non-counterexamples to S does not affect its manifest hypothesis.

By Lemma 3, the field F can be extended to another field E in which each of the coefficients c_i is a square. We define a function $\tau : X \to E^t$ by the rule:

$$\tau(\mathbf{a}) = (\sqrt{c_1}\pi_{\mathbf{t}_1}(\mathbf{a}), \ldots, \sqrt{c_t}\pi_{\mathbf{t}_t}(\mathbf{a}))$$

where $\sqrt{c_i}$ is some element whose square equals c_i. Informally, $\tau(\mathbf{a})$ encodes the terms of f that are satisfied by $\mathbf{a}$: if $\mathbf{t}_i \leq \mathbf{a}$ (so that the corresponding term is satisfied), then the ith component of $\tau(\mathbf{a})$ is $\sqrt{c_i}$; otherwise, this component is 0. Note that the value $f(\mathbf{a})$ can be recovered from $\tau(\mathbf{a})$ simply by taking its inner product with itself since

$$\tau(\mathbf{a}) \cdot \tau(\mathbf{a}) = \sum_{i=1}^{t} c_i \pi_{\mathbf{t}_i}(\mathbf{a}) = f(\mathbf{a}). \tag{6}$$

More generally,

$$\tau(\mathbf{a}) \cdot \tau(\mathbf{b}) = \sum_{i=1}^{t} c_i \pi_{\mathbf{t}_i}(\mathbf{a} \wedge \mathbf{b}) = f(\mathbf{a} \wedge \mathbf{b}) \tag{7}$$

since $\pi_{\mathbf{t}}(\mathbf{a}) \cdot \pi_{\mathbf{t}}(\mathbf{b}) = \pi_{\mathbf{t}}(\mathbf{a} \wedge \mathbf{b})$ for any $\mathbf{t}$, by definition of the meet operator.

Finally, we define the function $\gamma : S \to E^t$. This function is defined recursively for $\mathbf{a} \in S$ by the rule:

$$\gamma(\mathbf{a}) = \tau(\mathbf{a}) - \sum_{\substack{\mathbf{a}' \in terms(S) \\ \mathbf{a}' < \mathbf{a}}} \gamma(\mathbf{a}'). \tag{8}$$

Notice that γ's definition mimics the definition of the coefficients $\tilde{h}(\mathbf{a})$ of $hyp(S)$ given in equation (4). Here, $\tau(\mathbf{a})$ has taken the role of $f(\mathbf{a})$, and $\gamma(\mathbf{a})$ has taken the role of $\tilde{h}(\mathbf{a})$. In fact, just as we saw that $\tau(\mathbf{a}) \cdot \tau(\mathbf{a}) = f(\mathbf{a})$, so it will also turn out to be the case that $\gamma(\mathbf{a}) \cdot \gamma(\mathbf{a}) = \tilde{h}(\mathbf{a})$ for all $\mathbf{a} \in S$. Moreover, if $\mathbf{a}$ and $\mathbf{b}$ are distinct elements in $terms(S)$, we will see that their images under γ are perpendicular, i.e., $\gamma(\mathbf{a}) \cdot \gamma(\mathbf{b}) = 0$. These properties, which we prove in the next lemma, will allow us to apply Lemma 4 to complete the proof of the theorem.

Lemma 5 *For all $\mathbf{a}, \mathbf{b} \in terms(S)$, the following hold:*

1. $\gamma(\mathbf{a}) \cdot \gamma(\mathbf{a}) \neq 0$.
2. *If $\mathbf{a} \neq \mathbf{b}$ then $\gamma(\mathbf{a}) \cdot \gamma(\mathbf{b}) = 0$.*

Proof: We argue first that, for $\mathbf{a}, \mathbf{b} \in S$, if $\mathbf{a} \leq \mathbf{b}$ then

$$\gamma(\mathbf{a}) \cdot \tau(\mathbf{b}) = \tilde{h}(\mathbf{a}). \tag{9}$$

For fixed $\mathbf{b}$, this follows by an induction argument on S in which we prove the property for all of the elements of S in any order that is compatible with $\leq$. In other words,

when proving that the property holds for some element $\mathbf{a} \in S$, we assume inductively that it holds for all $\mathbf{a}' < \mathbf{a}$.

Suppose $\mathbf{a} \leq \mathbf{b}$. Then, taking the inner product of $\tau(\mathbf{b})$ with both sides of equation (8), we obtain

$$\gamma(\mathbf{a}) \cdot \tau(\mathbf{b}) = \tau(\mathbf{a}) \cdot \tau(\mathbf{b}) - \sum_{\substack{\mathbf{a}' \in terms(S) \\ \mathbf{a}' < \mathbf{a}}} \gamma(\mathbf{a}') \cdot \tau(\mathbf{b}).$$

If $\mathbf{a}'$ is as in the sum above, then $\gamma(\mathbf{a}') \cdot \tau(\mathbf{b}) = \tilde{h}(\mathbf{a}')$ by inductive hypothesis. Also, by equation (7), we have that $\tau(\mathbf{a}) \cdot \tau(\mathbf{b}) = f(\mathbf{a})$ since $\mathbf{a} \leq \mathbf{b}$. Thus,

$$\gamma(\mathbf{a}) \cdot \tau(\mathbf{b}) = f(\mathbf{a}) - \sum_{\substack{\mathbf{a}' \in terms(S) \\ \mathbf{a}' < \mathbf{a}}} \tilde{h}(\mathbf{a}')$$

and the claim follows by definition of $\tilde{h}(\mathbf{a})$.

To complete the lemma, we show that the following claims hold for all pairs $(\mathbf{a}, \mathbf{b}) \in S^2$:

1. if $\mathbf{a} = \mathbf{b}$ then $\gamma(\mathbf{a}) \cdot \gamma(\mathbf{a}) = \tilde{h}(\mathbf{a})$;
2. if $\mathbf{b} < \mathbf{a}$ and $\mathbf{b} \in terms(S)$ then $\gamma(\mathbf{a}) \cdot \gamma(\mathbf{b}) = 0$ (and symmetrically if $\mathbf{a} < \mathbf{b}$);
3. if $\mathbf{a} \not\leq \mathbf{b}$, $\mathbf{b} \not\leq \mathbf{a}$ and $\mathbf{a}, \mathbf{b} \in terms(S)$ then $\gamma(\mathbf{a}) \cdot \gamma(\mathbf{b}) = 0$.

These statements clearly imply the lemma: If $\mathbf{a} \in terms(S)$, then, by claim 1, $\gamma(\mathbf{a}) \cdot \gamma(\mathbf{a}) = \tilde{h}(\mathbf{a}) \neq 0$, and if $\mathbf{a}$ and $\mathbf{b}$ are distinct elements of $terms(S)$, then $\gamma(\mathbf{a}) \cdot \gamma(\mathbf{b}) = 0$ by claims 2 and 3.

The proof of these claims is by induction on S^2 using any order compatible with the partial order $\leq$, where it is understood that $(\mathbf{a}', \mathbf{b}') \leq (\mathbf{a}, \mathbf{b})$ if and only if $\mathbf{a}' \leq \mathbf{a}$ and $\mathbf{b}' \leq \mathbf{b}$.

Let $(\mathbf{a}, \mathbf{b}) \in S^2$. We assume inductively that the three claims hold for all pairs $(\mathbf{a}', \mathbf{b}') < (\mathbf{a}, \mathbf{b})$.

Proof of claim 1: Suppose $\mathbf{a} = \mathbf{b}$. By taking inner product of $\gamma(\mathbf{a})$ with both sides of equation (8), we see that

$$\gamma(\mathbf{a}) \cdot \gamma(\mathbf{a}) = \gamma(\mathbf{a}) \cdot \tau(\mathbf{a}) - \sum_{\substack{\mathbf{a}' \in terms(S) \\ \mathbf{a}' < \mathbf{a}}} \gamma(\mathbf{a}) \cdot \gamma(\mathbf{a}').$$

Note that, if $\mathbf{a}'$ is as in the sum above, then $\gamma(\mathbf{a}) \cdot \gamma(\mathbf{a}') = 0$ by claim 2 since $(\mathbf{a}, \mathbf{a}') < (\mathbf{a}, \mathbf{a})$. Thus,

$$\gamma(\mathbf{a}) \cdot \gamma(\mathbf{a}) = \gamma(\mathbf{a}) \cdot \tau(\mathbf{a}) = \tilde{h}(\mathbf{a})$$

by equation (9).

Proof of claim 2: Suppose $\mathbf{b} < \mathbf{a}$ and $\mathbf{b} \in terms(S)$. Similar to the proof above, we take the inner product of $\gamma(\mathbf{b})$ with both sides of equation (8) to obtain

$$\gamma(\mathbf{b}) \cdot \gamma(\mathbf{a}) = \gamma(\mathbf{b}) \cdot \tau(\mathbf{a}) - \sum_{\substack{\mathbf{a}' \in terms(S) \\ \mathbf{a}' < \mathbf{a}}} \gamma(\mathbf{b}) \cdot \gamma(\mathbf{a}').$$

As before, if $\mathbf{b} \neq \mathbf{a}'$ then $\gamma(\mathbf{b}) \cdot \gamma(\mathbf{a}') = 0$ by claims 2 and 3 (since $(\mathbf{a}', \mathbf{b}) < (\mathbf{a}, \mathbf{b})$). Thus,

$$\gamma(\mathbf{b}) \cdot \gamma(\mathbf{a}) = \gamma(\mathbf{b}) \cdot \tau(\mathbf{a}) - \gamma(\mathbf{b}) \cdot \gamma(\mathbf{b}) = 0$$

since $\gamma(\mathbf{b}) \cdot \tau(\mathbf{a}) = \gamma(\mathbf{b}) \cdot \gamma(\mathbf{b}) = \tilde{h}(\mathbf{b})$ by equation (9) and claim 1.

Proof of claim 3: Suppose $\mathbf{a} \not\leq \mathbf{b}$, $\mathbf{b} \not\leq \mathbf{a}$ and $\mathbf{a}, \mathbf{b} \in terms(S)$. By definition of $\gamma(\mathbf{a})$, we have

$$\tau(\mathbf{a}) = \gamma(\mathbf{a}) + \sum_{\substack{\mathbf{a}' \in terms(S) \\ \mathbf{a}' < \mathbf{a}}} \gamma(\mathbf{a}')$$

and similarly for $\tau(\mathbf{b})$. Thus,

$$\begin{aligned} \tau(\mathbf{a}) \cdot \tau(\mathbf{b}) &= \gamma(\mathbf{a}) \cdot \gamma(\mathbf{b}) + \sum_{\substack{\mathbf{a}' \in terms(S) \\ \mathbf{a}' < \mathbf{a}}} \gamma(\mathbf{a}') \cdot \gamma(\mathbf{b}) \\ &\quad + \sum_{\substack{\mathbf{b}' \in terms(S) \\ \mathbf{b}' < \mathbf{b}}} \gamma(\mathbf{a}) \cdot \gamma(\mathbf{b}') + \sum_{\substack{\mathbf{a}', \mathbf{b}' \in terms(S) \\ \mathbf{a}' < \mathbf{a}, \mathbf{b}' < \mathbf{b}}} \gamma(\mathbf{a}') \cdot \gamma(\mathbf{b}'). \end{aligned}$$

If $\mathbf{a}', \mathbf{b}'$ are as above, then $\gamma(\mathbf{a}') \cdot \gamma(\mathbf{b}) = 0$ by claims 2 and 3 since $(\mathbf{a}', \mathbf{b}) < (\mathbf{a}, \mathbf{b})$ and since $\mathbf{a}' \neq \mathbf{b}$ (otherwise, $\mathbf{b} \leq \mathbf{a}$). Similarly, $\gamma(\mathbf{a}) \cdot \gamma(\mathbf{b}') = 0$, and $\gamma(\mathbf{a}') \cdot \gamma(\mathbf{b}') = 0$ whenever $\mathbf{a}' \neq \mathbf{b}'$. Thus,

$$\tau(\mathbf{a}) \cdot \tau(\mathbf{b}) = \gamma(\mathbf{a}) \cdot \gamma(\mathbf{b}) + \sum_{\substack{\mathbf{c} \in terms(S) \\ \mathbf{c} \leq \mathbf{a} \wedge \mathbf{b}}} \gamma(\mathbf{c}) \cdot \gamma(\mathbf{c}).$$

That is,

$$\begin{aligned} \gamma(\mathbf{a}) \cdot \gamma(\mathbf{b}) &= \tau(\mathbf{a}) \cdot \tau(\mathbf{b}) - \sum_{\substack{\mathbf{c} \in terms(S) \\ \mathbf{c} \leq \mathbf{a} \wedge \mathbf{b}}} \gamma(\mathbf{c}) \cdot \gamma(\mathbf{c}) \\ &= \tau(\mathbf{a}) \cdot \tau(\mathbf{b}) - \gamma(\mathbf{a} \wedge \mathbf{b}) \cdot \gamma(\mathbf{a} \wedge \mathbf{b}) - \sum_{\substack{\mathbf{c} \in terms(S) \\ \mathbf{c} < \mathbf{a} \wedge \mathbf{b}}} \gamma(\mathbf{c}) \cdot \gamma(\mathbf{c}) \\ &= f(\mathbf{a} \wedge \mathbf{b}) - \tilde{h}(\mathbf{a} \wedge \mathbf{b}) - \sum_{\substack{\mathbf{c} \in terms(S) \\ \mathbf{c} < \mathbf{a} \wedge \mathbf{b}}} \tilde{h}(\mathbf{c}) = 0. \end{aligned}$$

The first equality can be seen as follows: if $\mathbf{a} \wedge \mathbf{b} \in terms(S)$, then the equality is trivial. Otherwise, since S is properly stable, $\mathbf{a} \wedge \mathbf{b} \in S - terms(S)$ so, by claim 1, $\gamma(\mathbf{a} \wedge \mathbf{b}) \cdot \gamma(\mathbf{a} \wedge \mathbf{b}) = \tilde{h}(\mathbf{a} \wedge \mathbf{b}) = 0$ (since $(\mathbf{a} \wedge \mathbf{b}, \mathbf{a} \wedge \mathbf{b}) < (\mathbf{a}, \mathbf{b})$), and the equality again is trivial. The second equality follows from equation (7) and claim 1, and the final equality follows from the definition of $\tilde{h}(\mathbf{a} \wedge \mathbf{b})$.

This completes the induction and the proof of the lemma. ■

Lemma 5 clearly implies that γ is injective on the restricted domain $terms(S)$, and moreover, that the set

$$\gamma(terms(S)) = \{\gamma(\mathbf{a}) : \mathbf{a} \in terms(S)\}$$

satisfies the hypotheses of Lemma 4. Thus, $|terms(S)| = |\gamma(terms(S))| \leq t$. ■

The next theorem will be helpful in proving that our algorithm is guaranteed to make progress on each iteration. Informally, it states that if S is properly stable, then adding elements to S can only increase the sparcity of its manifest hypothesis. Note that this property does not hold in general for unstable sets, or even for sets that are stable but not properly stable. (To see that Theorem 6 fails for sets that are not properly stable sets, let $X = \{0,1\}^4$, $F = \mathrm{GF}(2)$, and $f(\mathbf{x}) = x_1 + x_2 + x_3 + x_4$. Then the set $S = \{1110, 1101, 1011\}$ is stable (but not properly stable), and $terms(S) = S$; however, if $S' = S \cup \{1000\}$, then $terms(S') = \{1000\}$.)

Theorem 6 *Let $f : X \to F$ where F is a field and X is a semilattice. Let $S \subseteq X$ be finite and properly stable with respect to f. Let $S' \subseteq X$ be a finite superset of S. Then $|terms_f(S)| \leq |terms_f(S')|$.*

Proof: Let $h = hyp_f(S')$. Then, by Lemma 1, h is consistent with f on $S \subseteq S'$. Thus, $hyp_f(S) = hyp_h(S)$ and S is properly stable with respect to h. Therefore, by Theorem 2, $|terms_f(S)| = |terms_h(S)|$ is at most the sparcity of h, which is exactly $|terms_f(S')|$. ■

5 Analysis and Correctness

Using the theorems proved in the last section, we are now ready to fully analyze our algorithm.

For a set $S \subseteq X$ and integer r, we denote by $S^{\leq r}$ the set of elements in S of height at most r: $S^{\leq r} = \{\mathbf{a} \in S : ||\mathbf{a}|| \leq r\}$.

We begin by proving the essential properties of subroutine EasyCounterexample.

Lemma 7 *Let S_i and S_f be the initial and final values of program variable S on a call to* EasyCounterexample*, and let $\mathbf{c}$ be the value returned by the subroutine. Then the following hold:*

1. *If S_i is stable then $\mathbf{c}$ =* "STABLE" *and S_f is properly stable.*
2. *If S_i is not stable, then $\mathbf{c}$ is a counterexample for $hyp(S_\mathrm{f})$ and $S_\mathrm{f}^{\leq ||\mathbf{c}||}$ is properly stable.*

3. *The number of membership queries made during the execution of* EASYCOUNTEREXAMPLE *is at most* $\binom{t+1}{2}$.

Proof:

Part 1: If S_i is stable, then EASYCOUNTEREXAMPLE tests every pair $\mathbf{a}, \mathbf{b} \in terms(S_i)$ and discovers that for none of these is $\mathbf{a} \wedge \mathbf{b}$ a counterexample. Thus, for each such pair is $\mathbf{a} \wedge \mathbf{b}$ added to S. It follows that EASYCOUNTEREXAMPLE returns the flag "STABLE", and moreover that S_f is properly stable.

Part 2: If S_i is not stable, then for some pair $\mathbf{a}, \mathbf{b} \in terms(S_i)$, $\mathbf{a} \wedge \mathbf{b}$ is a counterexample so the subroutine returns a counterexample $\mathbf{c}$ rather than the flag "STABLE". Suppose that $\mathbf{a}$, $\mathbf{b}$ and T are as in the subroutine at the point at which $\mathbf{c} = \mathbf{a} \wedge \mathbf{b}$ is returned. To see that ${S_f}^{\leq ||\mathbf{c}||}$ is properly stable, consider a pair $\mathbf{a}', \mathbf{b}' \in terms({S_f}^{\leq ||\mathbf{c}||}) \subseteq terms(S_f)$. Since $\mathbf{c}$ is a counterexample, $\mathbf{c} \neq \mathbf{a}$ (by Lemma 1) so $\max(||\mathbf{a}'||, ||\mathbf{b}'||) \leq ||\mathbf{c}|| < ||\mathbf{a}||$. Thus, $\mathbf{a}', \mathbf{b}' \in T$ because of the greedy order in which elements are tested and added to T. Since EASYCOUNTEREXAMPLE did not halt when $\mathbf{a}', \mathbf{b}'$ were tested, $\mathbf{a}' \wedge \mathbf{b}'$ cannot be a counterexample. Thus, $\mathbf{a}' \wedge \mathbf{b}'$ was added to S_f at line 9, and therefore, $\mathbf{a}' \wedge \mathbf{b}' \in {S_f}^{\leq ||\mathbf{c}||}$. Hence, ${S_f}^{\leq ||\mathbf{c}||}$ is properly stable.

To prove the bound given in part 3 on the number of membership queries, it suffices to show that $|T| \leq t$ at all times. If S_i is stable, then $T \subseteq terms(S_i)$ so, by Theorem 2, $|T| \leq |terms(S_i)| \leq t$. If S_i is not stable, then $T \subseteq terms({S_f}^{\leq ||\mathbf{c}||})$ and by part 2 of this Lemma, ${S_f}^{\leq ||\mathbf{c}||}$ is stable; thus, again applying Theorem 2, $|T| \leq |terms({S_f}^{\leq ||\mathbf{c}||})| \leq t$. ■

Next, we prove some of the basic properties of ADDELEMENT.

Lemma 8 *Let S_i and S_f be the initial and final values of program variable S on a call to* ADDELEMENT *from* LEARNPOLY*; let $\mathbf{c}_i$ and $\mathbf{c}_f$ be similarly defined for $\mathbf{c}$. Then the following hold:*

1. *$\mathbf{c}_i$ is a counterexample for $hyp(S_i)$, and the set ${S_i}^{\leq ||\mathbf{c}_i||}$ is properly stable;*
2. *the set ${S_f}^{\leq ||\mathbf{c}_f||}$ is properly stable;*
3. *at most t membership queries are made during the entire execution of the subroutine.*

Proof: When ADDELEMENT was called from LEARNPOLY, it was passed a parameter $\mathbf{c}_i$ that was obtained either from EQUIV or from EASYCOUNTEREXAMPLE. In the latter case, part 1 of the lemma follows immediately from part 2 of Lemma 7. In the former case, the result follows from the assumed properties of the equivalence oracle, and from the fact that S is properly stable prior to each call to EQUIV (by part 1 of Lemma 7).

For part 2, consider the behavior of the algorithm after the program variable $\mathbf{c}$ has been set to $\mathbf{c}_f$; this setting either holds initially (if $\mathbf{c}_i = \mathbf{c}_f$), or occurs at some later point after executing line 6. In either case, on the next iteration of the outer loop (lines 2–10), $\ell = ||\mathbf{c}_f||$, and, on each succeeding execution of the loop, ℓ is decremented (since $\mathbf{c}$ never changes again). Thus, for each $\mathbf{a} \in terms(S_i)$ with $||\mathbf{a}|| \leq ||\mathbf{c}_f||$, $\mathbf{a} \wedge \mathbf{c}_f$

is at some point tested and found not to be a counterexample. Using part 1 of this lemma, and since $||\mathbf{c}_\mathrm{f}|| \leq ||\mathbf{c}_\mathrm{i}||$, it follows that $(S_\mathrm{i} \cup \{\mathbf{c}_\mathrm{f}\})^{\leq ||\mathbf{c}_\mathrm{f}||}$ is stable, and therefore that ${S_\mathrm{f}}^{\leq ||\mathbf{c}_\mathrm{f}||}$ is properly stable.

Part 3 follows immediately from Theorem 2 since ${S_\mathrm{i}}^{\leq ||\mathbf{c}_\mathrm{i}||}$ is stable and since each element of $terms({S_\mathrm{i}}^{\leq ||\mathbf{c}_\mathrm{i}||})$ is tested at most once. (Note that ℓ decreases on each iteration of the outer loop since if $\mathbf{a} \wedge \mathbf{c}$ is a counterexample as at line 6 then $\mathbf{a} \wedge \mathbf{c} \notin S_\mathrm{i}$ so $||\mathbf{a} \wedge \mathbf{c}|| < ||\mathbf{a}|| = \ell$.) ■

Next, we use the preceding results to show that the subroutine AddElement is called at most $nt+1$ times, where n is the height of X (i.e., the number of variables if X is the boolean lattice). This fact will allow us immediately to bound the number of equivalence queries made, and will be helpful for bounding the number of membership queries.

Lemma 9 *Given access to a target t-sparse polynomial f on a height-n semilattice, algorithm* LearnPoly *halts after executing subroutine* AddElement *at most $nt+1$ times.*

Proof: We prove this lemma by showing that whenever we add a counterexample $\mathbf{c}$ to S at line 11 of AddElementwe increase the number of hypothesis terms of height at or below $||\mathbf{c}||$ (i.e., the set $terms(S^{\leq ||\mathbf{c}||})$ strictly increases in size).

More formally, let $U = \{1, \ldots, n\} \times \{1, \ldots, t\} \cup \{(0,1)\}$. We describe below a procedure for "marking" elements of U. Initially, all elements of U are unmarked. We show that exactly one element is marked on each execution of AddElement, and we also show that no element is ever marked twice. Thus, AddElement is executed at most $|U| = nt + 1$ times.

Specifically, immediately following AddElement's execution, we "mark" element $p(\mathbf{c}, S) = (||\mathbf{c}||, |terms(S^{\leq ||\mathbf{c}||})|)$ where $\mathbf{c}$ and S are as given in the subroutine. (This marking is not actually performed by the algorithm — we use it merely as an aid in proving the theorem.)

First, we show that $p(\mathbf{c}, S)$ is actually an element of U. By part 2 of Lemma 8, the set $S^{\leq ||\mathbf{c}||}$ is properly stable. Thus, by Theorem 2, $||terms(S^{\leq ||\mathbf{c}||})|| \leq t$. So if $||\mathbf{c}|| > 0$ then $p(\mathbf{c}, S)$ is indeed an element of U (i.e., $1 \leq ||\mathbf{c}|| \leq n$ and $1 \leq ||terms(S^{\leq ||\mathbf{c}||})|| \leq t$). If $||\mathbf{c}|| = 0$ then $\mathbf{c} = \bot$ and $terms(S^{\leq ||\mathbf{c}||}) = \{\mathbf{c}\}$ so $p(\mathbf{c}, S) = (0, 1)$.

It remains to show that no element of U is marked twice. Suppose to the contrary that the same element is marked following two separate calls to AddElement, once when $\mathbf{c} = \mathbf{c}_1$ and $S = S_1$, and again later when $\mathbf{c} = \mathbf{c}_2$ and $S = S_2$. That is, $p(\mathbf{c}_1, S_1) = p(\mathbf{c}_2, S_2)$. Let $r = ||\mathbf{c}_1|| = ||\mathbf{c}_2||$.

Note that $\mathbf{c}_2 \in terms(S_2)$ since $\mathbf{c}_2$ was the last counterexample added to S_2 and by the way we compute $hyp(S_2)$. Note also that ${S_1}^{\leq r} \subseteq {S_2}^{\leq r} - \{\mathbf{c}_2\}$ since the algorithm never deletes elements from S.

Thus, by Theorem 6,

$$|terms({S_1}^{\leq r})| \leq |terms({S_2}^{\leq r} - \{\mathbf{c}_2\})|.$$

Since $\mathbf{c}_2$ is not less than any element in ${S_2}^{\leq r}$, it follows from the manner in which the manifest hypothesis is computed that $terms({S_2}^{\leq r} - \{\mathbf{c}_2\}) = terms({S_2}^{\leq r}) - \{\mathbf{c}_2\}$.

Thus, since $\mathbf{c}_2 \in terms(S_2)$,

$$|terms(S_1{}^{\leq r})| \leq |terms(S_2{}^{\leq r}) - \{\mathbf{c}_2\}| = |terms(S_2{}^{\leq r})| - 1 < |terms(S_2{}^{\leq r})|.$$

This contradicts that $p(\mathbf{c}_1, S_1) = p(\mathbf{c}_2, S_2)$. ■

Finally, we are ready to prove Theorem 10, the main result of this paper:

Theorem 10 *Given access to equivalence and membership queries for a target t-sparse polynomial f over a field F on a height-n semilattice X, the algorithm* LearnPoly *halts and outputs a hypothesis equivalent to f in polynomial time after making at most $nt+2$ equivalence queries and $(nt+1)(t^2+3t)/2$ membership queries.*

Proof: As is typically the case for equivalence-query algorithms, the procedure is automatically correct (in the sense that it outputs a hypothesis equivalent to the target) if it can be shown to halt after a bounded number of queries.

Since AddElement is executed at least once following each unsuccessful equivalence query, by Lemma 9, the number of equivalence queries is at most $nt+2$. Also, EasyCounterexample is executed exactly once following each execution of AddElement. Thus, combining Lemmas 7, 8 and 9, we see that the number of membership queries is at most $(nt+1)(t+\binom{t+1}{2}) = (nt+1)(t^2+3t)/2$. Finally, it is straightforward to verify using the lemmas developed above that the algorithm runs in polynomial time. ■

Setting $X = \{0,1\}^n$ we obtain the following immediate theorem.

Theorem 11 *Given access to equivalence and membership queries for a target t-sparse polynomial f on n boolean variables over a field F, the algorithm* LearnPoly *halts and outputs a hypothesis equivalent to f in polynomial time after making at most $nt+2$ equivalence queries and $(nt+1)(t^2+3t)/2$ membership queries.*

6 Applications and Extensions

In this section, we describe a number of applications and extensions of our main result.

Multilinear polynomials. We begin by showing that our algorithm can be used as a subroutine for learning multilinear polynomials when the domain of each variable is the entire field F, rather than $\{0,1\}$.

To prove this, it suffices to show that any counterexample in F^n can be used to derive another counterexample in $\{0,1\}^n$.

Let f be the target multilinear polynomial, and let h be a hypothesis. Let $d = f - h$. Suppose $\mathbf{c}$ is a counterexample to h and that there exists some i for which $c_i \notin \{0,1\}$. We partially evaluate d by fixing all the variables x_j to be c_j for $j \neq i$. We thus obtain the univariate polynomial $d'(x_i) = ax_i + b$ for some $a, b \in F$. Clearly, if $d'(0) = d'(1) = 0$ then $a = b = 0$ contradicting that $d'(c_i) \neq 0$. Therefore, there exists $y \in \{0,1\}$ such that $d'(y) \neq 0$. Thus the vector $\mathbf{c}'$ obtained from $\mathbf{c}$ by replacing c_i with y is a counterexample to h, and such a $\mathbf{c}'$ can be found with at most two membership

queries. Repeating this process at most n times we produce a counterexample in $\{0,1\}^n$.

Thus we have proved the following theorem.

Theorem 12 *Given access to equivalence and membership queries for a target t-sparse, n-variable multilinear polynomial $f : F^n \to F$ over a field F, there exists an algorithm that halts and outputs a hypothesis equivalent to f in polynomial time.*

Infinitely many attributes. Our main algorithm was shown to be effective for learning polynomials defined on any semilattice of finite height. The prime example of such a domain is of course the boolean lattice $\{0,1\}^n$. Here is another example: Let A be an infinite set, and let $\mathcal{A} = \{B \subseteq A : |B| \leq n\}$ be the collection of all subsets of A of cardinality at most n. Let the collection $\mathcal{A}$ be partially ordered by inclusion (i.e., $B \leq C$ if and only if $B \subseteq C$). Then $\mathcal{A}$ is a semilattice of height n, so our algorithm can be applied to efficiently learn a polynomial defined on it.

In fact, if we regard the set A as a collection of "attributes," then it becomes clear that a polynomial on this semilattice is just an ordinary polynomial defined over an infinite collection of variables (corresponding to the attributes in A), and a sample point (i.e., an element of $\mathcal{A}$) consists of a set of at most n attributes that hold for that example. In other words, the problem of learning a polynomial on this semilattice is exactly the problem of learning an ordinary polynomial in Blum's "infinite attribute" model [4], and so we obtain as a corollary to our main result that polynomials can be exactly identified in the infinite attribute model with membership and equivalence queries. Thus, although Blum gives a general technique for converting a "finite attribute" algorithm into one in the infinite attribute model, we obtain this result for polynomials by the direct argument given above.

Semilattices with infinite chains. Recall that our proof of the correctness and efficiency of our algorithm required that the target polynomial be defined on a semilattice of finite height. The finite-height requirement is, in general, necessary in order to achieve exact identification. For instance, the real interval $[0,1]$ forms a semilattice under the usual ordering, but no element of this set (except 0) has finite height. It is not hard to see that an adversarial oracle for equivalence and membership queries can force the learning algorithm to make an infinite number of queries, even if the target polynomial f is known to consist of a single term with coefficient 1 (so that, for some $a \in [0,1]$, $f(x)$ is 1 if $x \geq a$ and 0 otherwise).

However, if our goal is simply to *approximate* the target polynomial, then we can, in many situations, modify our algorithm to handle polynomials defined on a semilattice X that is not necessarily of finite height. More specifically, let $f : X \to F$ be the target polynomial of sparcity t, and let $S \subseteq X$ be a finite sample labeled by f. We claim that, given access to a membership oracle for f, our algorithm can be used to efficiently construct a hypothesis polynomial h of sparcity at most t that is consistent with f on S. The running time of the procedure is polynomial in t and $|S|$ (and any other parameters that may be relevant to the particular problem at hand).

To see that this is so, let X' be the subsemilattice of X obtained by closing the set S under the meet operation; that is, X' consists of all elements of X which are

the meet of a subset of the elements of S. It is not hard to show then that X' is a finite semilattice of height at most $|S|$. Further, the target polynomial f can be replaced by a polynomial $f' : X' \to F$ whose terms are all in X' and that equals f on all elements of X'. (Specifically, f' can be derived from f by replacing each term $\pi_{\mathbf{t}}$ by $\pi_{\mathbf{t}'}$ where $\mathbf{t}'$ is the meet of the set $\{\mathbf{a} \in X' : \mathbf{t} \leq \mathbf{a}\}$.) For our simulation, we regard f' as the target polynomial, X' as the target semilattice, and we simulate the equivalence oracle by responding to each equivalence query with any element of S on which f (or, equivalently, f') disagrees with the conjectured hypothesis (or with the "EQUIVALENT" flag if no such element exists). By the arguments above, and by Theorem 10, this simulation will produce a hypothesis of sparcity at most t that is consistent with f on S in time polynomial in t and $|S|$.

In many situations, such an algorithm for efficiently finding a "small" hypothesis consistent with a given sample is sufficient to guarantee efficient PAC-learnability. For example, if $X = \mathbb{R}^m$ is partially ordered by domination (so that $\mathbf{a} \leq \mathbf{b}$ if and only if $a_i \leq b_i$ for $i = 1, \ldots, m$), and if F is, say, $\mathbb{R}$ or GF(2), then a uniform convergence argument, such as those given by Blumer et al. [6] and Haussler [14], implies that a "small" hypothesis consistent with a randomly chosen sample will, with high probability, be a good approximation of the target function. Thus, by the arguments above, such polynomials can be efficiently PAC-learned from random examples given access to a membership oracle.

Functions representable by sparse polynomials. Finally, returning to the boolean lattice, we make some remarks on the sorts of functions that can be represented by sparse polynomials.

As noted briefly in Section 2, every function $g : \{0,1\}^\ell \to F$ can be represented by a 2^ℓ-sparse polynomial. Call such a function *ℓ-arbitrary*. Then clearly any ℓ-arbitrary function can be learned by our algorithm in time polynomial in 2^ℓ. More generally, we can replace each variable of g with a monomial $\mathbf{x}^{\mathbf{a}}$ for arbitrary $\mathbf{a} \in \{0,1\}^n$. The resulting function $g(\mathbf{x}^{\mathbf{a}_1}, \ldots, \mathbf{x}^{\mathbf{a}_\ell})$, when "multiplied out," can be represented by a 2^ℓ-sparse polynomial over $x_1, \ldots, x_n$. Generalizing further, we see that the ℓ-arbitrary functions can be added together to obtain a function of the form

$$\sum_{i=1}^{t} g_i(\mathbf{x}^{\mathbf{a}_{i1}}, \ldots, \mathbf{x}^{\mathbf{a}_{i\ell}})$$

which can be represented by a $t2^\ell$-sparse polynomial on n boolean variables. Thus, such functions can be learned in time polynomial in n, t and 2^ℓ.

As a specific example of this technique, we can show that logarithmic-depth decision trees can be learned in polynomial time: For each leaf i, let $p_i(\mathbf{x})$ be the value of the leaf node if it is reached on input $\mathbf{x}$, and 0 otherwise. The function computed by the decision tree is then $\sum_{i=1}^{t} p_i(\mathbf{x})$ where t is the number of leaves in the tree. Since $p_i(\mathbf{x})$ can be viewed as an ℓ-arbitrary function on the ℓ variables occurring along the path to leaf i, this shows that the computed function can be represented by a $t2^\ell$-sparse polynomial. Thus, the decision tree can be exactly identified in polynomial time if its depth ℓ is logarithmic.

The same result holds if each node's decision function is replaced by an arbitrary (monotone) monomial, rather than a single variable. Thus, we have shown that

logarithmic-depth decision trees in which each node is decided by a monomial can be exactly identified using equivalence and membership queries. Although it was known that this was possible for (ordinary) logarithmic-depth decision trees by the results of Kushilevitz and Mansour [17] and Bshouty [7], it appears that this result could not have been derived using previous methods for the case in which each decision node is a monomial.

7 Conclusions and Open Problems

We have shown that sparse polynomials over a field F defined on several boolean variables can be exactly identified using membership and equivalence queries. We have argued that this result depends largely on the lattice structure of the boolean domain, and that the result holds for a more generalized notion of polynomial defined on an arbitrary semilattice with no infinite chains. Among our extensions is a proof that our algorithm can be used to efficiently learn multilinear polynomials when the domain of each variable is all of F.

There are many open problems and possible directions for future research. For starters, we would like to know if our algorithm can be made robust to handle noise or errors in the data it is receiving. In this regard, we have some preliminary results which indicate that the algorithm can be modified to handle a small but significant level of random misclassification noise. (See also the related work of Ar et al. [2] and Gemmell and Sudan [12].)

Our algorithm is only able to learn functions that can be represented *exactly* by sparse polynomials. The algorithm would be much more practical if we could extend it to handle functions that can only be *approximated* by a sparse polynomial. This is an important open problem.

It would also be quite interesting to extend our algorithm to learn polynomials in which each "term" is a conjunction of literals, some of which are negated. For instance, for GF(2), this is the problem of learning a boolean formula that consists of an XOR of several monomials, each of which is a conjunction of negated or unnegated variables (rather than exclusively unnegated variables as was considered in this paper).

A similar problem is that of learning arbitrary polynomials (not necessarily multilinear) on the unrestricted domain F^n. In this regard, we have some preliminary results indicating that this is possible when each term of the target polynomial includes a limited number of variables of degree greater than one.

References

[1] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75:87–106, November 1987.

[2] Sigal Ar, Richard J. Lipton, Ronitt Rubinfeld, and Madhu Sudan. Reconstructing algebraic functions from mixed data. In *33rd Annual Symposium on Foundations of Computer Science*, pages 503–512, October 1992.

[3] Michael Ben-Or and Prasoon Tiwari. A deterministic algorithm for sparse multivariate polynomial interpolation. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, pages 301–309, May 1988.

[4] Avrim Blum. Learning boolean functions in an infinite attribute space. *Machine Learning*, 9(4):373–386, 1992.

[5] Avrim Blum and Mona Singh. Learning functions of k terms. In *Proceedings of the Third Annual Workshop on Computational Learning Theory*, pages 144–153, August 1990.

[6] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the Association for Computing Machinery*, 36(4):929–965, October 1989.

[7] Nader H. Bshouty. Exact learning via the monotone theory. In *34th Annual Symposium on Foundations of Computer Science*, November 1993.

[8] Michael Clausen, Andreas Dress, Johannes Grabmeier, and Marek Karpinski. On zero-testing and interpolation of k-sparse multivariate polynomials over finite fields. *Theoretical Computer Science*, 84:151–164, 1991.

[9] Andreas Dress and Johannes Grabmeier. The interpolation problem for k-sparse polynomials and character sums. *Advances in Applied Mathematics*, 12:57–75, 1991.

[10] A. Dür and J. Grabmeier. Applying coding theory to sparse interpolation. *SIAM Journal on Computing*, 22(4):695–704, August 1993.

[11] Paul Fischer and Hans Ulrich Simon. On learning ring-sum-expansions. *SIAM Journal on Computing*, 21(1):181–192, February 1992.

[12] Peter Gemmell and Madhu Sudan. Highly resilient correctors for polynomials. *Information Processing Letters*, 43(4):169–174, September 1992.

[13] Dima Yu. Grigoriev, Marek Karpinski, and Michael F. Singers. Fast parallel algorithms for sparse multivariate polynomial interpolation over finite fields. *SIAM Journal on Computing*, 19(6):1059–1963, December 1990.

[14] David Haussler. Decision theoretic generalizations of the PAC model for neural net and other learning applications. *Information and Computation*, 100(1):78–150, 1992.

[15] Lisa Hellerstein and Manfred Warmuth. Interpolating GF[2] polynomials. Unpublished manuscript.

[16] I. N. Herstein. *Topics in Algebra*. Wiley, second edition, 1975.

[17] Eyal Kushilevitz and Yishay Mansour. Learning decision trees using the Fourier spectrum. *SIAM Journal on Computing*, 22(6):1331–1348, 1993.

[18] Yishay Mansour. Randomized interpolation and approximation of sparse polynomials. In *Automata, Languages and Programming: 19th International Colloquium*, pages 261–272, July 1992.

[19] Ron M. Roth and Gyora M. Benedek. Interpolation and approximation of sparse multivariate polynomials over GF(2). *SIAM Journal on Computing*, 20(2):291–314, April 1991.

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

[21] Herbert S. Wilf. Hadamard determinants, Möbius functions, and the chromatic number of a graph. *Bulletin of the American Mathematical Society*, 74(5):960–964, September 1968.

[22] Richard Zippel. Probabilistic algorithms for sparse polynomials. In *Symbolic and Algebraic Computation*, pages 216–226. Springer-Verlag, June 1979.

[23] Richard Zippel. Interpolating polynomials from their values. *Journal of Symbolic Computation*, 9:375–403, 1990.

Appendix

In this appendix, we present an alternative proof of Theorem 2 based on the computation of the rank of certain matrices. Recall that the *rank* of a matrix A is equal to the maximum number of linearly independent columns of the matrix A.

Alternative Proof of Theorem 2: It suffices to prove the theorem in the case that X is finite. For if X is infinite, then we can replace X by X', the finite subsemilattice that is generated by closing the finite set

$$S \cup \{\mathbf{a} : \tilde{f}(\mathbf{a}) \neq 0\}$$

under the meet operation. Clearly, if Theorem 2 holds for X', then it holds for X as well. Thus, we assume henceforth without loss of generality that X is finite.

The high level idea of this proof is to construct two matrices G and $\hat{G}$ for which we can argue that:

1. $\text{rank}(G) = t$, the sparcity of f;
2. $\text{rank}(\hat{G}) = |terms(S)|$, the sparcity of $hyp(S)$; and
3. $\text{rank}(\hat{G}) \leq \text{rank}(G)$.

Obviously, these three facts together suffice to prove the theorem. The method of constructing these matrices is inspired by a technique used by Wilf [21].

Let the $r = |X|$ elements of X be indexed $\mathbf{a}_1, \ldots, \mathbf{a}_r$ in a manner consistent with the partial ordering $\leq$, i.e., in such a way that if $i > j$ then $\mathbf{a}_i \not\leq \mathbf{a}_j$. We can then represent the partial order $\leq$ by an $r \times r$ matrix $Z = (z_{ij})$ where

$$z_{ij} = \begin{cases} 1 & \text{if } \mathbf{a}_i \leq \mathbf{a}_j \\ 0 & \text{otherwise.} \end{cases}$$

Then Z is upper triangular (i.e., $z_{ij} = 0$ if $i > j$) and all diagonal entries z_{ii} are equal to 1. Thus, Z has determinant 1, and so it is nonsingular.

Next, let $D = (d_{ij})$ be the $r \times r$ diagonal matrix whose diagonal elements are given by the coefficients of f. That is,

$$d_{ij} = \begin{cases} \tilde{f}(\mathbf{a}_i) & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases}$$

Note that the rank of D is equal to the number of non-zero diagonal entries, which is exactly t, the sparcity of f.

To complete the construction of G, we finally let $G = (g_{ij}) = Z^T D Z$. Since Z is nonsingular,

$$\text{rank}(G) = \text{rank}(D) = t. \tag{10}$$

Also, we can explicitly compute each entry of G as follows:

$$g_{ij} = \sum_{k,\ell} z_{ki} d_{k\ell} z_{\ell j} = \sum_{k} z_{ki} z_{kj} \tilde{f}(\mathbf{a}_k).$$

Note that

$$z_{ki} z_{kj} = \begin{cases} 1 & \text{if } \mathbf{a}_k \leq \mathbf{a}_i \wedge \mathbf{a}_j \\ 0 & \text{otherwise.} \end{cases}$$

Thus,

$$g_{ij} = \sum_{\substack{\mathbf{a} \in X \\ \mathbf{a} \leq \mathbf{a}_i \wedge \mathbf{a}_j}} \tilde{f}(\mathbf{a}) = f(\mathbf{a}_i \wedge \mathbf{a}_j).$$

We use a similar construction for the matrix $\hat{G}$. First, let

$$T = \{i : \mathbf{a}_i \in \mathit{terms}(S)\} = \{t_1, \ldots, t_s\}$$

where $t_1 < \cdots < t_s$, and $s = |\mathit{terms}(S)|$. Let $\hat{Z} = (\hat{z}_{ij})$ be the $T \times T$ submatrix of Z (i.e., $\hat{z}_{ij} = z_{t_i t_j}$). Then the same argument used above shows that $\hat{Z}$ is also nonsingular.

Next, we define $\hat{D} = (\hat{d}_{ij})$ to be the $s \times s$ diagonal matrix whose diagonal elements are the coefficients of $h = \mathit{hyp}(S)$. That is,

$$\hat{d}_{ij} = \begin{cases} \tilde{h}(\mathbf{a}_{t_i}) & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases}$$

Since each diagonal element is non-zero, $\hat{D}$ has full rank s.

Finally, we let $\hat{G} = (\hat{g}_{ij}) = \hat{Z}^T \hat{D} \hat{Z}$. As before, because $\hat{Z}$ is nonsingular,

$$\text{rank}(\hat{G}) = \text{rank}(\hat{D}) = s = |terms(S)|. \tag{11}$$

Moreover, we can compute the entries of $\hat{G}$ explicitly as before to obtain:

$$\hat{g}_{ij} = \sum_{\substack{\mathbf{a} \in terms(S) \\ \mathbf{a} \leq \mathbf{a}_{t_i} \wedge \mathbf{a}_{t_j}}} \tilde{h}(\mathbf{a}) = h(\mathbf{a}_{t_i} \wedge \mathbf{a}_{t_j}) = f(\mathbf{a}_{t_i} \wedge \mathbf{a}_{t_j})$$

where the last equality follows from our assumption of stability.

Thus, the matrix $\hat{G}$ is exactly the $T \times T$ submatrix of G, which implies that

$$\text{rank}(\hat{G}) \leq \text{rank}(G). \tag{12}$$

The theorem follows immediately from Equations (10), (11) and (12). ■