Learning Definite Horn Formulas from Closure Queries

Marta Arias[a,1,3,*], José L. Balcázar[a,1,3], Cristina Tîrnăucă[b,1,2]

[a]*LARCA Research Group, Department of Computer Science,*
Universitat Politècnica de Catalunya, Barcelona, Spain
[b]*Departamento de Matemáticas, Estadística y Computación,*
Universidad de Cantabria, Santander, Spain

Abstract

A definite Horn theory is a set of n-dimensional Boolean vectors whose characteristic function is expressible as a definite Horn formula, that is, as conjunction of definite Horn clauses. The class of definite Horn theories is known to be learnable under different query learning settings, such as learning from membership and equivalence queries or learning from entailment. We propose yet a different type of query: the closure query. Closure queries are a natural extension of membership queries and also a variant, appropriate in the context of definite Horn formulas, of the so-called correction queries. We present an algorithm that learns conjunctions of definite Horn clauses in polynomial time, using closure and equivalence queries, and show how it relates to the canonical Guigues-Duquenne basis for implicational systems. We also show how the different query models mentioned relate to each other by either showing full-fledged reductions by means of query simulation (where possible), or by showing their connections in the context of particular algorithms that use them for learning definite Horn formulas.

*Corresponding author

 Email addresses: `marias@cs.upc.edu` (Marta Arias),
`jose.luis.balcazar@upc.edu` (José L. Balcázar), `cristina.tirnauca@unican.es`
(Cristina Tîrnăucă)

1. Introduction

Query Learning [1] is one of the most well-known and studied theoretical learning models available. According to this model, a learning agent or algorithm interacts with the world by asking questions (that is, *queries*) to one or several oracles that reveal partial information about an unknown but previously fixed concept (the *target*). Learning is achieved when the learning agent outputs a representation of the unknown concept. Naturally, as it is customary in a computational context, the number of queries that the algorithms need to make in order to learn is a resource that one needs to account for and restrict; efficient algorithms in the Query Learning context should address to the oracle a polynomial amount of queries. Finally, a concept class is said to be *learnable* within a query learning model if there exists an algorithm that, by asking a polynomial number of its available queries, is able to discover any previously fixed concept from the concept class.

Query Learning thus deals with determining learnability of different concept classes under different variants of learning models (that is, with different query types). The more complex a concept class is, the more queries will in general be necessary to learn it. And one concept class may be learnable under a particular set of query types but not others (see e.g. [2] for examples of this). Several extensions, of different focus and generality, appeared subsequently. One very general notion of query learning is that of [3]; in a less general level, some of these different extensions are of interest for this paper.

In this paper, we focus our attention to the class of (propositional) definite Horn formulas. The seminal paper by Angluin, Frazier and Pitt [4] established that this class is indeed query learnable[4]. The types of queries that they allowed are the most commonly used in query learning [1]: standard membership queries (SMQs) and standard equivalence queries (SEQs). In an SMQ, the algorithm asks whether a particular example (a truth assignment in our case) belongs to the unknown target concept, and the answer it receives is YES if the assignment is satisfied and NO otherwise. In an SEQ, the algorithm asks whether a particular Horn formula (the hypothesis) is semantically equivalent to the target formula, and the answer it receives

---

[4]To be more precise, their algorithm learns the more general class of Horn formulas.

2

is YES if it is indeed so and NO otherwise. In the case of a negative answer, a *counterexample* is provided, that is, an assignment that is satisfied by the target but not by the hypothesis or vice versa. The counterexample is then used by the algorithm to refine its hypothesis and learning continues.

Since the introduction of the first query learning models [1], several further variants of query types have been introduced and studied. One is the family of Unspecified Attribute Value queries [5], the query-learning parallel to the Restricted Focus of Attention extension to the PAC model [6]. Its key traits are the context of n-dimensional Boolean vectors (for fixed n) and the ability to handle each dimension somehow "individually", by means of the use of "don't-care" symbols to allow the query to focus on specific dimensions.

A second variant, also working on n-dimensional Boolean vectors but moving to a slightly more abstract notion of query, is proposed in [7]: the entailment query, where simple formulas (in that concrete case, Horn clauses) play the role of individual examples. In this protocol, in an entailment membership query (EMQ) the learner proposes a Horn clause and receives, as answer, a Boolean value indicating whether it is entailed by the target, seen as a propositional (in that case, Horn) formula. Similarly, in the entailment equivalence query (EEQ), a Horn formula is proposed and, in case of a negative answer, the provided counterexample is a Horn clause that is entailed by exactly one of the two formulas, the target and the query.

Yet a different variant of entailment query is employed in [8] for the algorithm known as Attribute Exploration, also in a context very similar to learning Horn clauses. This is a protocol where the query is an implication, that is, a conjunction of clauses sharing the same antecedent; the main difference is as follows: one gets as answer either YES if the implication is entailed by the target, or a counterexample that satisfies the target but not the implication. Thus, this variant is midway through between plain entailment membership, to which it resembles most, and standard equivalence, because a counterexample assignment is received in the negative case.

Finally, in [9, 10, 11, 12], we find a different extension: Correction Queries, which model a very intuitive idea from linguistics: instead of a simple NO answer as in the case of SMQs, the teacher provides a "correction", that is, an element of the target language at minimum distance from the queried example.

One must note that, whereas several positive results prove that the availability of certain query combinations allows for polynomial-time learnability, there are negative results that show that many representation classes are im-

possible to learn from polynomially many individual queries like membership or equivalence [1, 2].

This paper's contributions are two-fold. First, we propose a quite natural notion of *closure query* (CQ) and a polynomial time learning algorithm for definite Horn Boolean functions via equivalence and closure queries; it is closely related to the algorithms in [4] and [7]. The second part of the paper studies the relationships between our newly introduced closure query model and other well-known query models. While some of the relationships are already known (and these are duly noted), we obtain interesting new ones. More precisely, our algorithm yields back, through a quite intuitive transformation, the algorithm for Learning from Entailment [7]. Additionally, as we shall see as well, also the celebrated algorithm to learn definite Horn theories from membership and equivalence queries of [4] can be related to this approach, in that the usage it makes of positive examples can be understood as progressing towards the identification of unavailable closures. We believe that these connections we develop provide insight, and also help in establishing an overview of the strengths and weaknesses of all the variants under study. In addition to the relation of the learning algorithm proposed in the first half of the paper to these already existing variants, further relationships between models are shown to be possible in the more general form of query simulation.

Closure queries share a number of traits with each of the query models discussed before. Under a natural notion of "correcting upwards" (see below for precise definitions), in the context of a definite Horn target, the closest correction to a negative example is exactly its *closure* under the target. Thus, we find a variant of correction query for definite Horn targets that allows for (limited) manipulation of individual dimensions of the Boolean hypercube, as do the other query models we have mentioned, and provides an explanation of "what are we looking for" along both the processes of entailment queries and the positive examples of the learning algorithm from membership and equivalence queries. Our advances are based on the novel view on Horn learning via queries deployed more recently in [13, 14]. The introduction of the closure query is, in fact, motivated by these two papers in the following sense: the main aim in revisiting the original algorithm [4] was to improve on its query complexity. This objective is still unfulfilled; however, we believe that the closure query provides the "right amount" of expressiveness and information to the algorithm in order to capture the essence of difficulty in learning, while at the same time making some of the book-keeping details

4

easier to deal with. Thus, working with the closure query model, we believe that we are in a better position to answer the fundamental question of whether the original algorithms of [4, 7] are indeed optimal or can otherwise be improved upon.

2. Preliminaries

We work within the standard framework in propositional logic, where one is given an indexable set of propositional variables of cardinality n, Boolean functions are subsets of the Boolean hypercube $\{0, 1\}^n$, and these functions are represented by logical formulas over the variable set in the standard way. Binary strings of length n assign a Boolean value for each variable, and are therefore called assignments; given any Boolean function or formula H, the fact that assignment x makes it true (or "satisfies" it) is denoted $x \models H$. Following the standard overloading of the operator, $H \models H'$ means that, for every assignment x, if $x \models H$ then $x \models H'$. Assignments are partially ordered bitwise according to $0 \leq 1$ (the usual partial order of the hypercube); the notation is $x \leq y$.

A literal is a variable or its negation. A conjunction of literals is a *term*, and if none of the literals appears negated it is a *positive term*, also often referred to as a *monotone term* or *monotone conjunction*. We often identify positive terms and mere sets of variables; in fact, we switch back and forth between set-based notation and assignments. We denote variables with letters from the beginning of the alphabet $(a, b, c, ..)$, terms, or equivalently subsets of variables, with Greek letters $(\alpha, \beta, ..)$ and assignments with letters from the end of the alphabet $(x, y, z, ..)$. We may abuse notation at times and it should be understood that if we use a subset α when an assignment is expected, it is to be interpreted as the assignment that sets to 1 exactly those variables in α. We denote this explicitly when necessary by $x = [\alpha]$. Similarly, if we use an assignment x where a subset of variables is expected, it is to be understood that we mean the set of variables that are set to 1 in x. We denote this explicitly by $\alpha = [x]$. Clearly, we have a bijection between sets of propositional variables and assignments, and $x = [[x]]$ and $\alpha = [[\alpha]]$ for all assignments x and variable sets α.

2.1. Horn Logic

In this paper we are only concerned with definite Horn functions, and their representations using conjunctive normal form (CNF). A Horn CNF

formula is a conjunction of Horn clauses. A clause is a disjunction of literals. A clause is *definite Horn* if it contains exactly one positive literal, and it is *negative* if all its literals are negative. A clause is *Horn* if it is either definite Horn or negative.

Horn clauses are generally viewed as implications where the negative literals form the antecedent of the implication (a positive term), and the singleton consisting of the positive literal, if it exists, forms the consequent of the clause. As just indicated, along this paper it will always exist.

An implication $\alpha \rightarrow \beta$, where both α and β are sets of propositional variables with α possibly empty, but not β, is to be interpreted as the conjunction of definite Horn clauses $\bigwedge_{b \in \beta} \alpha \rightarrow b$. A semantically equivalent interpretation is to see both sets of variables α and β as positive terms; the Horn formula in its standard form is obtained by distributivity over the variables of β. Of course, any result that holds for Horn formulas in implicational form with no other restrictions also holds for the clausal representation unless it explicitly depends of the implications proper, such as counting the number of implications, as we will do below. Furthermore, we often use sets to denote conjunctions, as we do with positive terms, also at other levels: a generic (implicational) CNF $\bigwedge_i (\alpha_i \rightarrow \beta_i)$ is often denoted in this text by $\{(\alpha_i \rightarrow \beta_i)\}_i$. Parentheses are mostly optional and generally used for ease of reading.

An assignment $x \in \{0,1\}^n$ satisfies the implication $\alpha \rightarrow \beta$, denoted $x \models \alpha \rightarrow \beta$, if it either falsifies the antecedent or satisfies the consequent, that is, $x \not\models \alpha$ or $x \models \beta$ respectively, where now we are interpreting both α and β as positive terms ($x \models \alpha$ if and only if $\alpha \subseteq [x]$ if and only if $[\alpha] \leq x$, see Lemma 1 of [14]).

Not all Boolean functions are Horn. The following semantic characterization is a well-known classic result of [15, 16], proved in the context of propositional Horn logic e.g. in [17]:

Theorem 1 ([15, 16, 17]). *A Boolean function admits a Horn CNF representation if and only if the set of assignments that satisfy it is closed under bit-wise intersection.*

A Horn function admits several syntactically different Horn CNF representations; in this case, we say that these representations are equivalent. Such representations are also known as *bases* for the Boolean function they represent. The *size* of a Horn function is the minimum number of clauses that a Horn CNF representing it must have. The *implication size* of a Horn

function is defined analogously, but allowing formulas to have implications instead of clauses. Clearly, every definite clause can be phrased as an implication, and thus the implication size of a given Horn function is always at most that of its standard size as measured in the number of clauses.

Horn CNF representations may as well include unnecessary implications. We will need to take this into account: an implication or clause in a Horn CNF H is *redundant* if it can be removed from H without changing the Horn function represented. A Horn CNF is *irredundant* or *irreducible* if it does not contain any redundant implication or clause. Notice that an irredundant H may still contain other sorts of redundancies, such as implications with consequents larger than strictly necessary.

2.2. Closure Operator and Equivalence Classes

We will employ the well-known method of *forward chaining* for definite Horn functions; see e.g. [18]. Given a definite Horn CNF $H = \{\alpha_i \to \beta_i\}_i$ and an initial subset of propositional variables α, we can construct a chain of subsets of propositional variables by successively adding right-hand sides of implications, provided that the corresponding left-hand side is already contained in the current subset. Given a set of variables α, the maximal outcome of this process is denoted $\alpha^\star$, and contains all the variables "implied" by the set of variables α. As it is well-known, $\alpha^\star$ is well-defined, and only depends on the Boolean function represented by H, not on the representation H itself. The corresponding process on assignments provides the analogous operator $x^\star$.

Note that the closure operator is defined with respect to a function which is not explicitly included in the notation $x^\star$. It should be clear from the text, however, with respect to what function the closure is taken.

It is easy to see that the $\star$ operator is extensive (that is, $x \le x^\star$ and $\alpha \subseteq \alpha^\star$), monotonic (if $x \le y$ then $x^\star \le y^\star$, and if $\alpha \subseteq \beta$ then $\alpha^\star \subseteq \beta^\star$) and idempotent ($x^{\star\star} = x^\star$, and $\alpha^{\star\star} = \alpha^\star$) for all assignments x, y and variable sets α, β; that is, $\star$ is a closure operator. Thus, we refer to $x^\star$ as the *closure* of x w.r.t. a definite Horn function f. An assignment x is said to be *closed* iff $x^\star = x$, and similarly for variable sets. The following holds for every definite Horn function f (see Theorem 3 in [14]):

Proposition 2 ([14])**.** *Let f be a definite Horn function; let α be an arbitrary variable subset, b any variable and x an arbitrary assignment. Then,*

   1. *$f \models \alpha \to b$ if and only if $b \in \alpha^\star$,*

2. $x = x^\star$ *if and only if* $x \models f$,

3. $x^\star = \wedge\{y \mid x \le y \text{ and } y \models f\}$,

Therefore, $f \models [y] \rightarrow [y^\star]$ whenever the closure of y is computed with respect to f. Moreover, for any assignment x, there is a uniquely defined assignment y evaluated positively by f with y bitwise minimal such that $x \le y$, namely, $y = x^\star$.

For a fixed function f, this closure operator induces a partition over the set of assignments $\{0,1\}^n$ in the following straightforward way: two assignments x and y belong to the same class if $x^\star = y^\star$, where both closures are taken w.r.t. f. This notion of equivalence class carries over as expected to the power set of propositional variables: the subsets α and β belong to the same class if $\alpha^\star = \beta^\star$. It is worth noting that each equivalence class consists of a possibly empty set of assignments that are not closed and a single closed set, its representative.

Furthermore, the notion of equivalence classes carries over to implications by identifying an implication with its antecedent. Thus, two implications belong to the same class if their antecedents have the same closure (w.r.t. a fixed f). Thus, the class of an implication $\alpha \rightarrow \beta$ is, essentially, $\alpha^\star$.

Example 1. *This example is taken from [19]. Let $H = \{e \rightarrow d, bc \rightarrow d, bd \rightarrow c, cd \rightarrow b, ad \rightarrow bce, ce \rightarrow ab\}$. Thus, the propositional variables are a, b, c, d, e, f. The following table illustrates the partition induced by the equivalence classes on the implications of H, where closures are taken with respect to H itself. The first column is the implication identifier, the second column is the implication itself, and the third column corresponds to the class of the implication. As one can see, there are three equivalence classes: one containing the first implication, another one containing implications 2, 3, and 4; and a final one containing implications 5 and 6.*

1	$e \rightarrow d$	*ed*
2	$bc \rightarrow d$	*bcd*
3	$bd \rightarrow c$	*bcd*
4	$cd \rightarrow b$	*bcd*
5	$ad \rightarrow bce$	*abcde*
6	$ce \rightarrow ab$	*abcde*

2.3. A Related Closure Operator

Now we proceed to define another important operator which is similar in flavor to the closure operator $\star$ seen above.

Let H be any definite Horn CNF, and α any variable subset. Let $H(\alpha)$ be those implications of H whose antecedents fall in the same equivalence class as α, namely, $H(\alpha) = \{\alpha_i \to \beta_i \mid \alpha_i \to \beta_i \in H$ and $\alpha^\star = \alpha_i^\star\}$. The closure is taken with respect to H itself.

Given a definite Horn CNF H and a variable subset α, we introduce a new operator $\bullet$ [19, 20, 21] that we define as follows: $\alpha^\bullet$ is the closure of α *with respect to the subset of implications $H \setminus H(\alpha)$*. That is, in order to compute $\alpha^\bullet$ one does forward chaining starting with α *but one is not allowed to use implications in $H(\alpha)$*.

Example 2. *Let $H = \{a \to b, a \to c, c \to d\}$. Then, $(ac)^\star = abcd$ but $(ac)^\bullet = acd$ since $H(ac) = \{a \to b, a \to c\}$ and we are only allowed to use the implication $c \to d$ when computing $(ac)^\bullet$.*

This new operator is, in fact, a closure operator, well-known in the field of Formal Concept Analysis; there, assignments that are closed with respect to it are sometimes called *quasi-closed*.

2.4. Saturation and the Guigues-Duquenne Basis

In this section we review briefly part of our results from our previous work [14]. We will skip many details as they can be found in the aforementioned article. These results are, in fact, an interpretation of the work of [19, 21] which were stated in the context of formal concepts, closure systems and lattices.

We say that an implication $\alpha \to \beta$ of a definite Horn CNF H is

- *left-saturated* if $\alpha = \alpha^\bullet$ (the quasi-closure is taken with respect to H)

- *right-saturated* if $\beta = \alpha^*$ (the closure is taken with respect to H)

- *saturated* if it is both left and right-saturated (the closure is taken with respect to H)

Then, a definite Horn CNF is saturated if all of its implications are. Moreover, any saturated definite Horn CNF must be irredundant (see Lemma 2 of [14] for a proof). A result from [19, 21] states that

Theorem 3 ([19, 21]). *Definite Horn functions have at most one saturated basis, which is of minimum implicational size. This basis is called the Guigues-Duquenne (GD) basis.*

9

See [14] for additional discussion. In particular, there we show that the GD basis of a given definite Horn CNF representation can be computed with the following procedure: using forward chaining, right-saturate every clause. Then, use forward chaining again to compute the left-saturation of the left hand sides of the implications using the information on the equivalence classes of the existing implications to do the left-saturation properly. Finally, remove all those clauses that are redundant.

2.5. Closure Queries

The above leads naturally to a clear notion of closure query (CQ):

Definition 1. *For a fixed definite Horn formula T on n propositional variables and given n-bit vector y, the answer to a closure query on input y is $y^\star$, that is, the closure of y with respect to T.*

This query can be seen as a natural variant of correction query: under the condition that all corrections are "upwards", namely, that they are allowed only to change a zero into a one, Proposition 2 tells us that, for every assignment y that is negative for T, there is a unique "closest" correction query, and it is exactly $y^\star$, the closure of y with respect to T.

Closure queries provide us with a way of correctly identifying right hand sides of implications in one shot, since saturated implications are always of the form $[y] \to [y^\star]$. We shall see in the next section that this is exploited during the learning process.

An oracle answering closure queries can be implemented to run in linear time in $|T|$ and n (see Theorem 2 of [14]).

3. Learning Definite Horn Theories from Closure and Equivalence Queries

As usual in Query Learning, a target definite Horn function T is fixed, and the learning algorithm interacts with an environment able to provide information about T in the form required by the corresponding query protocol. In our case, this amounts to the learning algorithm being able to use at any time the closure $y^\star$ of any Boolean vector y as necessary, as it can be obtained from a closure query (the closure is computed with respect to the target T). Equivalence queries (denoted as $EQ()$ in the algorithms) are used in the standard manner as control of termination: the algorithm finishes exactly when the equivalence query receives a positive answer, which guarantees correctness provided that the algorithm is shown to terminate.

Theorem 4. *Definite Horn formulas are learnable from equivalence and closure queries in polynomial time.*

Proof. Each equivalence query will take the form of a hypothesis definite Horn formula $hyp(N)$ based on a list of n-bit vectors N; namely, it will be a conjunction of implications, defined as follows:

$$hyp(N) = \bigwedge_{y \in N} [y] \to [y^\star]$$

In fact, as we shall see momentarily, N will be a list of negative examples, as usual in Horn clause learning. Clearly, given N, $hyp(N)$ can be constructed easily using closure queries.

  We observe now that the combination of an inequality and a closure leads to a membership query: z is a negative example if and only if $z < z^\star$, because always $z \leq z^\star$, and positive examples are exactly those that coincide with their closure, by Proposition 2.

  We combine these ingredients as described in Algorithm 1 which we will call *ClH*. The proof of its correctness is built out of the following two lemmas. These lemmas refer to the elements in N, the list of counterexamples, as $y_1, y_2, y_3, \ldots, y_{|N|}$.

Lemma 5. $T \models hyp(N)$, *therefore counterexamples are always negative.*

Proof of Lemma 5. Take any assignment y. Since the closure $y^\star$ is taken with respect to the theory T, we have that $T \models [y] \to [y^\star]$ for every y and in particular all those $y \in N$, and therefore, $T \models \bigwedge_{y \in N}[y] \to [y^\star] = hyp(N)$ as required. $\qquad\square$

Lemma 6. *For $i < j$, there is a positive z with $y_i \wedge y_j \leq z \leq y_j$, and, therefore, each y_i violates different implications of T.*

Proof of Lemma 6. We argue inductively along the successive updates of N. We need to establish the fact (1) at the time of appending a new element of N, and (2) we need to argue that refinements to existing $y \in N$ that take place maintain the fact stated. We will show (2) in detail; (1) is proven similarly and so we omit the details.

  First note the easiest case whereby y_i gets refined into $y_i' = y_i \wedge x$. This leaves y_j untouched, and brings down $y_i \wedge y_j$ into $y_i' \wedge y_j$; the same value of z, given by the induction hypothesis on y_i, y_j before the update, will do: $y_i' \wedge y_j \leq y_i \wedge y_j \leq z \leq y_j$.

Now consider the case in which y_j is refined into $y_j' = y_j \wedge x$. We assume as inductive hypothesis that a corresponding z exists before the refinement: $y_i \wedge y_j \le z \le y_j$.

We establish first the following auxiliary claim: there is a positive example z' for which $y_i \wedge x \le z' \le x$. To find such z', observe that y_i came before y_j but was not chosen for refinement; either $y_i \wedge x$ is itself positive, and we can simply choose $z' = y_i \wedge x$, or $y_i \le x$. Since x was a negative counterexample, it must satisfy the query, so we have that $x \models [y_i] \to [y_i^\star]$; therefore $y_i^\star \le x$ since $y_i \le x$. We pick $z' = y_i^\star$, which is of course positive.

At this point, we have the already existing z fulfilling $y_i \wedge y_j \le z \le y_j$, and the z' just explained for which $y_i \wedge x \le z' \le x$. Observe the following: $y_i \wedge y_j' = y_i \wedge y_j \wedge x = y_i \wedge y_i \wedge y_j \wedge x = (y_i \wedge y_j) \wedge (y_i \wedge x)$. The first half of this last expression is bounded above by z, and the second half is bounded above by z', therefore $y_i \wedge y_j' \le z \wedge z' \le y_j \wedge x = y_j'$. Moreover, both z and z' being positive, and the target being closed under intersection, ensures that $z \wedge z'$ is positive.

The induction basis case of appending a new x to N is handled in the same way: the positive z' obtained in the same manner fulfills directly the condition $y_i \wedge x \le z' \le x$, which is what we need.

Finally, the property that there exists a positive z s.t. $y_i \wedge y_j \le z \le y_j$ for every $i < j$ implies that each different y_i, y_j must falsify a different implication of the target T. Suppose otherwise by way of contradiction that both counterexamples are falsifying the same implication $\alpha \to \beta$. Then, we would have that $[\alpha] \le y_i$ and $[\alpha] \le y_j$ so that $[\alpha] \le y_i \wedge y_j \le z$. Since z is positive, then $z \models \alpha \to \beta$ and so $[\beta] \le z \le y_j$. Therefore $y_j \models \alpha \to \beta$ thus contradicting our assumption. $\qquad\square$

To prove termination, it suffices to note that by the previous lemma, N cannot be longer than the number of implications in the target. After each iteration, either an existing counterexample decreases in at least one bit (which can happen at most n times for each existing counterexample), or a new one is added (which can happen at most m times, where m is the implication size of T). Hence, the total number of equivalence queries issued is at most $nm + m + 1 = O(nm)$. As to the number of closure queries, in each iteration we need to issue at most m queries when checking intersections with existing members of N, which makes a total of $O(m^2 n)$ closure queries. Notice that we could store and avoid the queries needed for building the hypothesis $hyp(N)$ and therefore we do not need to account for the extra

m queries (which in any case does not affect the asymptotic of the query count). In terms of time, the outer loop is executed $O(mn)$ times, and each iteration has a cost of mn: the factor m is due to looping over all $y_i \in N$, and the factor n for the manipulations of vectors of length n, totaling a time complexity[5] of $O(m^2 n^2)$. □

Algorithm 1 Learning from Closures Algorithm *ClH*

 1: $N = [\,] \; //$ empty list
 2: **while** $EQ(hyp(N)) = (\text{NO}, x)$ **do**
 3:   $//$ we will show below that x is negative
 4:   **for** $y_i \in N$, in order **do**
 5:    $y = x \wedge y_i$
 6:    **if** $y < y_i$ and $y < y^\star$ **then**
 7:     $y_i = y$
 8:     **break**
 9:   **if** no y_i was changed **then**
10:    add x at the end of N

3.1. The Horn Formula Obtained

We prove now the main fact about algorithm *ClH*, characterizing its output. Most of the proof is discharged into the following technical lemma.

Lemma 7. *At the time of issuing the equivalence query, $hyp(N)$ is left-saturated.*

Proof. For $hyp(N)$ to be left-saturated it is enough to show that $y_i \models [y_j] \rightarrow [y_j^\star]$ whenever $i \neq j$ since this implies that $y_i = y_i^\bullet$ or equivalently y_i is closed with respect to $H \setminus H([y_i])$, where $H = hyp(N)$.

In order to show that an arbitrary $y_i \in N$ satisfies an arbitrary clause of $[y_j] \rightarrow [y_j^\star] \in hyp(N)$ whenever $i \neq j$, we proceed to show that $y_i \geq y_j$ implies $y_i \geq y_j^\star$ and so the implication is necessarily satisfied.

We assume, then, that $y_i \geq y_j$. If $i < j$, by Lemma 6 we know that $y_i \wedge y_j \leq z \leq y_j$, and so we have $y_j \leq z \leq y_j$ which is impossible since all y_j

---

are negative and z is positive. Therefore, it must be the case that $i > j$ and then Lemma 6 guarantees that $y_j \wedge y_i = y_j \leq z \leq y_i$. Monotonicity of the closure operator implies that $y_j^\star \leq z^\star$ and so $y_j^\star \leq z$ since $z^\star = z$. Finally, $y_j^\star \leq z \leq y_i$ implies $y_j^\star \leq y_i$ as required. □

Theorem 8. *The output of Algorithm ClH is the GD basis of the target.*

Proof. The output is the last hypothesis queried, which receives a positive answer. By the previous lemma, all the antecedents are left-saturated with respect to $hyp(N)$; but, as the answer is positive, $hyp(N)$ is equivalent to the target, hence all the antecedents are left-saturated with respect to the target. By construction, the right-hand sides of the queries are always closures under the target. Hence, the final query is a saturated definite Horn formula for the target. As we have indicated earlier, there is a single saturated definite Horn formula for any definite Horn theory: its GD basis. This is, therefore, the output of the algorithm. □

4. Relationships among Query Learning Models

This section attempts to clarify the relationships between our algorithm and the previously published versions that work under slightly different learning models [7, 4, 14]. The original AFP algorithm [4, 14] works under what we will refer to as the *Standard Query Model* which uses standard equivalence queries (SEQs) and standard membership queries (SMQs).

The algorithm LRN in [7] works under the *Entailment Query Model* which uses entailment membership queries (EMQs) and entailment equivalence queries (EEQs). Entailment queries are somewhat more sophisticate versions of the standard set-theoretic queries. In these queries, the role of assignments is played here by clauses. In the entailment setting, a membership query becomes a query to find out whether a concrete clause provided by the learner is entailed by the target. As in the set-theoretic setting, the equivalence query is a Horn formula, but the counterexample in case of nonequivalence is a clause that is entailed by exactly one of the two Horn formulas: the query and the target. This is, in fact, the major difference with set-theoretic queries: the entailment-based equivalence query does not return an n-bit vector but, instead, a clause.

Generally speaking, there are two ways in which the relation between these algorithms becomes apparent: the first one being that some queries can be directly simulated by others, and so algorithms are the product of

reductions; but, also, there may be a way to specifically run simulations of one particular algorithm within another, even if the query protocol does not allow for direct simulation.

This section is divided into two parts. The first part (Section 4.1) will show direct simulations of several types of queries by other query types. This type of reduction shows, in fact, the relationships among the three models considered (standard, entailment, and closure) independent of the algorithm employed. The second part (Section 4.2) shows executions of the actual algorithms that lead to similar behaviors in the sense of having identical evolution of intermediate hypotheses.

4.1. Query Simulation

In this section we discuss cases where queries of one type can be directly answered by (efficient) algorithms using another set of queries. In this case, an algorithm working under one model can be directly made to work under another model by using the appropriate query-answering algorithms as black boxes. These are, in fact, query model reductions.

In the following subsections we will detail several of these reductions. In some cases we will see how we can simulate one type of query by its analogue under another model; in other cases, we may need both types of queries (membership and equivalence, for example) to be able to simulate another query. In our presentation, T stands for the target Horn function.

4.1.1. Entailment Queries Simulate Closure, Standard Membership, and Standard Equivalence Queries

EMQ $\rightarrow$ CQ. It is not hard to answer a CQ when EMQs are available. Given y, to construct $y^\star$ (its closure with respect to T), we test, for each variable b not in $[y]$, whether $T \models [y] \rightarrow b$ by means of EMQs. We include in $[y^\star]$, apart from the variables that are already present in $[y]$, all the b's corresponding to positive answers from the EMQ. Clearly, this constructs $y^\star$ with a linear cost in terms of EMQs.

EMQ $\rightarrow$ SMQ. The same process provides for SMQs. Indeed, a membership query on an assignment x receives a positive answer if and only if $x = x^\star$, as per Proposition 2. Essentially, membership is negative if and only if there exists some variable b not in $[x]$ such that $T \models [x] \rightarrow b$. Again, the cost is linear.

15

EMQ+EEQ → SEQ. We should note that this case is just a detailed version of Footnote 4 in [7]. When answering an SEQ, unless the hypothesis is already equivalent to the target, we need to return an assignment that satisfies the target but not the hypothesis or vice versa. We first make an EEQ with the hypothesis and in return obtain a clause; from this clause we need to find an assignment that distinguishes the target from our hypothesis. We have two cases: it is a positive counterexample (entailed by the target but not by the hypothesis), or it is a negative counterexample (entailed by the hypothesis but not by the target).

The easier case is when the clause produced by the EEQ is positive. We transform it into a negative counterexample assignment x as follows. Let $\alpha \rightarrow b$ be the counterexample clause, so that T entails $\alpha \rightarrow b$ but $hyp(N)$ does not. There must be x that satisfies $hyp(N)$ but does not satisfy $\alpha \rightarrow b$, so that it cannot satisfy T because of the entailment from T. Such an x is what we want.

How do we actually find it? To fail $\alpha \rightarrow b$, it must satisfy α, and also all the consequences of α under $hyp(N)$ in order to satisfy $hyp(N)$. The closure of $[\alpha]$ under $hyp(N)$ (call it w) will do. Variable b is not in that closure because the variables in the closure of α under $hyp(N)$ are exactly those variables v for which $hyp(N)$ entails $\alpha \rightarrow v$, and for $v = b$ it is not the case. Hence, w fails $\alpha \rightarrow b$, which is entailed by T, so w cannot satisfy T, and satisfies $hyp(N)$ because it is a closure under it. So, in order to answer the EQ in this case one EEQ is enough and the time complexity is what it takes to do forward-chaining with the hypothesis, which can be done (when implemented carefully) in linear time in the number of implications in the hypothesis and the number of variables [22].

The remaining case (counterexample clause entailed by the hypothesis but not by the target) can in fact be handled in the same way. The only difference is that, instead of closing $[\alpha]$ under the hypothesis $hyp(N)$, we close it under the target, obtaining $[\alpha]^\star$ via the simulation of closures by EMQs. So, in this case, one EEQ and a linear number of EMQs are needed in the worst case.

As a consequence of the ability of entailment queries to implement both CQs and SEQs, from Theorems 4 and 8 we obtain:

Theorem 9. *The following statements hold.*

1. *[7] Horn theories are learnable from entailment queries in polynomial time.*

16

2. *Further, such learning can be done so as to output the GD basis of the target.*

4.1.2. Closure and Equivalence Queries Simulate Entailment

CQ → EMQ. A CQ can easily simulate a membership query of the entailment protocol. Given a clause $\alpha \to v$, we can find out whether the target entails it by just asking for the closure of the left-hand side and testing whether $v \in \alpha^\star$. One single CQ suffices.

CQ+SEQ → EEQ. For the simulation of an equivalence query of entailment, of course we resort to an SEQ; but we must transform the assignment we get as counterexample into a counterexample clause for entailment. Given a negative counterexample assignment x, use a CQ to obtain $x^\star \neq x$ and choose any variable v that is true in $x^\star$ but not in x. Then, our counterexample query is $[x] \to v$: as x is positive for the query, v is not a consequence of $[x]$ for the query, but it is with respect to the target, as $v \in [x^\star]$. Similarly, given a positive counterexample x, that is, therefore, negative for the query, we can find a counterexample clause $[x] \to v$ by finding some $v \notin [x]$ that follows by forward chaining from $[x]$ using the hypothesis in the query. Besides the SEQ, we spend at most one additional CQ in this process. The total time would be $\mathcal{O}(nm)$ (here, m is the implication size of the hypothesis).

As a corollary, we obtain the following linear reductions among these three models:

Corollary 1. *The following statements hold.*

1. **CQ+SEQ ⟷ EMQ+EEQ.** *The entailment and closure learning models are equivalent (up to a linear number of queries).*
2. **EMQ+EEQ → SMQ+SEQ.** *Entailment can simulate the standard protocol (up to a linear number of queries).*
3. **CQ+SEQ → SMQ+SEQ.** *The closure protocol can simulate the standard protocol (up to a linear number of queries).*

It is worth noting that it is also possible to simulate closure queries (CQ) with the standard protocol (i.e., SEQ+SMQ → CQ) by means of the following trivial (polynomial-query) reduction: when asked to compute a closure, we invoke the AFP algorithm of [4] and once we discover the target we can easily compute the closure. Notice that this takes $O(nm^2)$ queries so a polynomial reduction is indeed possible; however, we would like to see strictly

better complexities. By transitivity, we would also obtain the (trivial) reduction SEQ+SMQ $\to$ EEQ+EMQ using the same trick. It remains an open question whether the reduction SEQ+SMQ $\to$ CQ can be done with better query complexity.

We can show, however, that having equivalence queries is necessary for the reduction to work. That is, if equivalence queries are not available, then the reduction SMQ $\to$ CQ is not possible with a polynomial number of queries, as the following theorem shows:

Theorem 10. *Answering a CQ may require an exponential number of SMQs.*

Proof. Let $\mathcal{F}$ be a family of Horn theories: $\mathcal{F} = \{f_x | x \in \{0, 1\}^n, x \neq 1^n\}$ where f_x is the conjunction of two parts:

$$f_x = \bigwedge_{v \in [x]} (\emptyset \to v) \wedge \bigwedge_{w \notin [x]} (w \to [1^n])$$

The first half of f_x guarantees that any satisfying assignment y is such that $x \leq y$, the second half guarantees that no assignment y such that $x < y < 1^n$ satisfies f_x. Thus, each f_x is satisfied by exactly two assignments: x itself and the top 1^n.

Now, we want to answer a CQ for the assignment 0^n. For an arbitrary target $f_x \in \mathcal{F}$, the answer should be x. But obviously we do not know what the target is and we need to answer the closure query by means of querying the standard membership query oracle. Answering the closure query correctly corresponds to identifying the target function f_x among all candidates in $\mathcal{F}$ (of which there are $2^n - 1$). We use an adversarial strategy to show the exponential lower bound: all the answers to any membership query are going to be negative unless the input assignment to the query is 1^n. Each query rules out only one potential target function and thus an exponential number of queries is needed. $\square$

In fact, Theorem 10 fits the general lower bounding scheme described in Lemma 2 of [1]. As a corollary we obtain that EMQs cannot be simulated with a polynomial number of SMQs either.

Corollary 2. *Answering an EMQ may require an exponential number of SMQs.*

Query simulation	Query complexity	Time complexity
SMQ $\to$ EMQ	$\mathcal{O}(2^n)$	$\mathcal{O}(2^n)$
SMQ $\to$ CQ	$\mathcal{O}(2^n)$	$\mathcal{O}(2^n)$
EMQ $\to$ CQ	$\mathcal{O}(n)$	$\mathcal{O}(n)$
EMQ $\to$ SMQ	$\mathcal{O}(n)$	$\mathcal{O}(n)$
CQ $\to$ EMQ	1	$\mathcal{O}(n)$
CQ $\to$ SMQ	1	$\mathcal{O}(n)$
SEQ+CQ $\to$ EEQ	1 SEQ + 1 CQ	$\mathcal{O}(nm)$
EEQ+EMQ $\to$ SEQ	1 EEQ + $\mathcal{O}(n)$ EMQ	$\mathcal{O}(nm)$

Table 1: Relationship between different queries. As it is customary, n stands for the number of propositional variables; m stands for the implicational size of the input hypothesis.

Algorithm	Query complexity	Time complexity
AFP	$\mathcal{O}(m^2n)$ SMQs & $\mathcal{O}(mn)$ SEQs	$\mathcal{O}(m^2n^2)$
LRN	$\mathcal{O}(m^2n)$ EMQs & $\mathcal{O}(mn)$ EEQs	$\mathcal{O}(m^2n^2)$
ClH	$\mathcal{O}(m^2n)$ CQs & $\mathcal{O}(mn)$ SEQs	$\mathcal{O}(m^2n^2)$

Table 2: Query complexity and time complexity for AFP, LRN and *ClH*.

Table 1 summarizes the results from this section.

Additionally, Table 2 summarizes the query and time complexities of the three algorithms that we compare in this paper for learning definite Horn theories. Note that while the worst-case complexities are the same for the three algorithms, there are cases where LRN and *ClH* are going to be running faster than AFP, due to their slightly more powerful query models.

4.2. Algorithm Run Simulation

In this section we deal with the two remaining cases in which, as far as we know, the queries are not directly simulable (that is, not without learning the target first). We show that the full runs "are", in the sense that a run of one algorithm is embedded in some run of the other.

Namely, each of the algorithms that we consider here, even on the same target, may exhibit different runs. More precisely, runs differ among them in which counterexamples are provided, and in which order.

4.2.1. AFP Runs that Mimic ClH Runs

In the original membership and equivalence queries protocol, the AFP algorithm is not guaranteed to receive only negative counterexamples. The reason is the lack of the closure query, that provides us with positive examples.

In fact, each run of *ClH* can be mimicked through a run of AFP as follows. Fix the run of *ClH* that receives the sequence of counterexamples $x_1, x_2, \ldots, x_k$. We construct inductively a specific run of AFP that will receive this sequence of negative counterexamples in the same order, plus positive ones as needed in between them. Consider the situation where it has just received the j-th of them, with $j = 0$ corresponding to the start of the algorithm. The refinement process is the same in both cases, where the tests for positive intersections are made via closures in one algorithm and through direct memberships in the other. However, at the point of constructing the query, one of the antecedents is either new, or newly refined. For the rest, inductively, the closures are the same as in the previous query, but AFP does not have available the closure of the changed antecedent in order to use it as consequent. It assumes the strongest possible consequent (or, in the variant in [14], the strongest consequent compatible with positive examples seen so far, so as to avoid the same counterexample to show up over and over). In the specific run we are constructing, let y_i be the newly obtained antecedent. AFP might happen to hit upon $y_i^\star$ on the basis of its available information: then, it is asking exactly the next query of the *ClH* run. Otherwise, it is proposing too large a consequent for y_i: then, we give AFP the positive counterexample $[y_i^\star]$, which fails the y_i clause yet is positive, because it is a closure. After this positive example, again the next query is exactly *ClH*'s query. In either case, AFP proceeds and gets the $j + 1$ negative counterexample.

Note that, along the way, a full formalization of this simulation (which we consider unnecessary, as the intuition is clearly conveyed) would provide an alternative proof of Theorem 8, as we get that every equivalence query (including its output) made by *ClH* is also a query made in some run of AFP on the same target, and it is proved in [14] that all queries of AFP are saturated.

Query complexity. Looking closely at the closure queries made by *ClH*, we note that they can be of two types: those that are issued to find out the "right consequent" of a newly added or updated implication in $hyp(N)$, and those that are issued to check whether the intersections of negative counterexam-

ples are positive or not (line 6 of Algorithm 1). If *ClH* makes a SEQs, then $a+b$ closure queries are made: a accounting for the former type (consequents of implications), and b for the latter type. So *ClH* issues a SEQs an $a+b$ CQs. Then, the corresponding run of AFP will make at most $2a$ SEQs (since we may need to feed all closures of antecedents as positive counterexamples in order to guide the algorithm towards the "correct consequents") and b SMQs corresponding to checking the "sign" of interesections in order to know what antecedent to update. So, both runs have the same total complexity, namely $2a + b$, although *ClH* uses extra CQs to save on SEQs.

Of course, our discussion so far only applies to specific runs of AFP constructed in that particular way. However, the properties of AFP proved in [14] show that all runs need to, eventually, identify the proper closures; in general, these necessary positive counterexamples may not come at the place we are placing them, but instead can come later; and, instead of the closures $[y_i^\star]$ that reduce the right-hand sides at once, we may reduce them one bit at a time through several positive counterexamples. However, in a somewhat loose sense, we can say that AFP is implementing the closure queries through potentially shuffled batches of positive counterexamples. This complicated procedure, whose most relevant property is actually the goal of obtaining closures, makes us expect that progress on the understanding of AFP, and, hopefully, either a proof of optimality or an improvement of its query complexity, could be obtained indirectly as a byproduct of the study of our simpler, but essentially equivalent, algorithm *ClH*.

4.2.2. AFP Runs that Mimic Entailment Runs of LRN

A similar development can be provided for mimicking runs of the Learning from Entailment algorithm. We refrain from getting into too much details here, as that would require, among other explanations, to review fully here the algorithm from [7]. However, for the benefit of the reader who knows, or plans to study soon, that algorithm, we briefly point out how the simulation goes; it is quite similar to the one in the previous subsection.

Fix the run that receives the sequence of counterexample clauses $\alpha_1 \to x_1$, $\alpha_2 \to x_2$, ..., $\alpha_k \to x_k$. We construct inductively a specific run of AFP that will receive a sequence of negative counterexamples, each corresponding, in a precise sense, to each of these clauses. More precisely, consider assignment w_i defined as the closure of $[\alpha_i]$ under the i-th hypothesis, to which $\alpha_i \to x_i$ itself is a counterexample. Being a closure under the hypothesis, w_i is positive for it; however, it does not have variable x_i set to 1, because the clause is

not entailed by the hypothesis, and this makes it a negative counterexample, because the clause *is* entailed by the target.

Again, each refinement process is identical in both algorithms, so the difference is again upon constructing the new query. The entailment algorithm can use entailment memberships to hit the correct right-hand side. Instead, we consider a run of AFP as before, where the appropriate positive counterexamples are provided right away to lead the algorithm to the correct next query of the simulated entailment run.

Again, we consider unnecessary to provide a full formalization of this simulation. However, since, again, all queries of AFP are saturated [14], we note that from such a full formalization we can obtain a slightly stronger version of Theorem 9: in fact, the LRN algorithm already constructs the GD basis of the target, because every equivalence query there, including its output, is also a query made in some run of AFP on the same target.

5. Related Open Problems

We would like to study how to extend our result to general Horn functions; the main difficulty being that it is not clear what closure means in the general case. In our previous work [14] we come up with a pwm-reduction [2] that makes learning possible for general Horn under the model of standard equivalence and standard membership queries. It is left for future work to study a new reduction that works under the closure query model.

It also remains to prove or disprove the following relationships:

- EMQ cannot be obtained with a linear number of SMQs and SEQs;

- CQ cannot be obtained with a linear number of SMQs and SEQs;

- EEQ cannot be obtained with a polynomial number of SEQ;

- SEQ cannot be obtained with a polynomial number of EEQ.

References

[1] D. Angluin, Queries and concept learning, Machine Learning 2 (4) (1987) 319–342.

[2] D. Angluin, M. Kharitonov, When won't membership queries help?, Journal of Computer and System Sciences 50 (2) (1995) 336–355.

[3] J. L. Balcázar, J. Castro, D. Guijarro, J. Köbler, W. Lindner, A general dimension for query learning, J. Comput. Syst. Sci. 73 (6) (2007) 924–940.

[4] D. Angluin, M. Frazier, L. Pitt, Learning conjunctions of Horn clauses, Machine Learning 9 (1992) 147–164.

[5] S. A. Goldman, S. Kwek, S. D. Scott, Learning from examples with unspecified attribute values, Inf. Comput. 180 (2) (2003) 82–100.

[6] S. Ben-David, E. Dichterman, Learning with restricted focus of attention, in: COLT, 1993, pp. 287–296.

[7] M. Frazier, L. Pitt, Learning from entailment: An application to propositional Horn sentences, in: ICML, Morgan Kaufmann, 1993, pp. 120–127.

[8] B. Ganter, Attribute exploration with background knowledge, Theor. Comput. Sci. 217 (2) (1999) 215–233.

[9] L. Becerra-Bonache, A. H. Dediu, C. Tirnăucă, Learning DFA from correction and equivalence queries, in: Y. Sakakibara, S. Kobayashi, K. Sato, T. Nishino, E. Tomita (Eds.), ICGI, Vol. 4201 of Lecture Notes in Computer Science, Springer, 2006, pp. 281–292.

[10] C. Tirnăucă, A note on the relationship between different types of correction queries, in: A. Clark, F. Coste, L. Miclet (Eds.), ICGI, Vol. 5278 of Lecture Notes in Computer Science, Springer, 2008, pp. 213–223.

[11] C. Tirnăucă, Language learning with correction queries, Ph.D. thesis, Rovira i Virgili University, Tarragona, Spain (2009).

[12] C. Tirnăucă, S. Kobayashi, Necessary and sufficient conditions for learning with correction queries, Theor. Comput. Sci. 410 (47-49) (2009) 5145–5157.

[13] M. Arias, J. L. Balcázar, Canonical Horn representations and query learning, in: R. Gavaldà, G. Lugosi, T. Zeugmann, S. Zilles (Eds.), ALT, Vol. 5809 of Lecture Notes in Computer Science, Springer, 2009, pp. 156–170.

[14] M. Arias, J. L. Balcázar, Construction and learnability of canonical Horn formulas, Machine Learning 85 (3) (2011) 273–297.

[15] A. Horn, On sentences which are true of direct unions of algebras, J. of Symbolic Logic 16 (1956) 14–21.

[16] J. McKinsey, The decision problem for some classes of sentences without quantifiers, J. Symbolic Logic 8 (1943) 61–76.

[17] R. Khardon, D. Roth, Reasoning with models, Artificial Intelligence 87 (1-2) (1996) 187 – 213.

[18] H. Kleine Büning, T. Lettmann, Propositional logic: deduction and algorithms, Cambridge University Press, 1999.

[19] J. Guigues, V. Duquenne, Familles minimales d'implications informatives resultants d'un tableau de données binaires, Math. Sci. Hum. 95 (1986) 5–18.

[20] D. Maier, Minimum covers in relational database model, J. ACM 27 (1980) 664–674.

[21] M. Wild, A theory of finite closure spaces based on implications, Advances in Mathematics 108 (1994) 118–139.

[22] W. F. Dowling, J. H. Gallier, Linear-time algorithms for testing the satisfiability of propositional Horn formulae, J. Log. Program. 1 (3) (1984) 267–284.