Learning and Verifying Quantified Boolean Queries by Example

Azza Abouzied*, Dana Angluin*, Christos Papadimitriou**,
Joseph M. Hellerstein**, Avi Silberschatz*
*Yale University, ** University of California, Berkeley
azza@cs.yale.edu, angluin@cs.yale.edu, christos@cs.berkeley.edu,
hellerstein@cs.berkeley.edu, avi@cs.yale.edu

ABSTRACT

To help a user specify and verify quantified queries — a class of database queries known to be very challenging for all but the most expert users — one can question the user on whether certain data objects are *answers* or *non-answers* to her intended query. In this paper, we analyze the number of questions needed to learn or verify *qhorn* queries, a special class of Boolean quantified queries whose underlying form is *conjunctions of quantified Horn expressions*. We provide optimal *polynomial-question* and *polynomial-time* learning and verification algorithms for two subclasses of the class qhorn with upper constant limits on a query's *causal density*.

Categories and Subject Descriptors

H.2.3 [**Database Management**]: Languages—*query languages*; I.2.2 [**Artificial Intelligence**]: Automatic Programming—*program synthesis, program verification*; I.2.6 [**Artificial Intelligence**]: Learning—*concept learning*

Keywords

quantified boolean queries, qhorn, query learning, query verification, example-driven synthesis

1. INTRODUCTION

It's a lovely morning, and you want to buy a box of chocolates for your research group. You walk into a chocolate store and ask for "a box with dark chocolates — some sugar-free with nuts or filling". However, your server is a pedantic logician who expects first-order logic statements. In response to your informal query he places in front of you a hundred boxes! Despite your frustration, you are intrigued: you open the first box only to find one dark, sugar-free chocolate with nuts and many other varieties of white chocolates that you didn't order. You push it aside, indicating your disapproval, and proceed to the second. Inside, you are wondering: Is there hope that I can communicate to this person my needs through a sequence of such interactions?

Everyday, we request things from each other using informal and incomplete query specifications. Our casual interactions facilitate such under-specified requests because we have developed questioning skills that help us clarify such requests. A typical interlocutor might ask you about corner cases, such as the presence of white chocolates in the box, to get to a precise query specification by example. As requesters, we prefer to begin with an outline of our query — the key properties of the chocolates — and then make our query precise using a few examples. As responders, we can build a precise query from the query outline and a few positive or negative examples — acceptable or unacceptable chocolate boxes.

Typical database query interfaces behave like our logician. SQL interfaces, for example, force us to formulate precise quantified queries from the get go. Users find quantified query specification extremely challenging [2, 13]. Such queries evaluate propositions over *sets of tuples* rather than individual tuples, to determine whether a set as a whole satisfies the query. Inherent in these queries are (i) the grouping of tuples into sets, and (ii) the binding of query expressions with either existential or universal quantifiers. Existential quantifiers ensure that some tuple in the set satisfies the expression, while universal quantifiers ensure that all tuples in the set satisfy the expression.

To simplify the specification of quantified queries, we built DataPlay [2]. DataPlay tries to mimic casual human interactions: users first specify the simple propositions of a query. DataPlay then generates a simple quantified query that contains all the propositions. Since, this query may be incorrect, users can label query results as *answers* or *non-answers* to their intended query. DataPlay uses this feedback on example tuple-sets to fix the incorrect query. Our evaluation of DataPlay shows that users prefer example-driven query specification techniques for specifying complex quantified queries [2]. Motivated by these findings, we set out to answer the question: *How far can we push the example-driven query specification paradigm?* This paper studies the theoretical limits of using examples to *learn* and to *verify* a special subclass of quantified queries, which we call *qhorn*, in the hope of eventually making query interfaces more human-like.

1.1 Our contributions

We formalize a query learning model (§2) where users specify propositions that form the building blocks of a Boolean quantified query. A learning algorithm then asks the users *membership questions*: each question is an example data object, which the user classifies as either an *answer* or a *non-answer*. After a few questions, the learning algorithm terminates with the unique query that satisfies the user's

PODS'13, June 22–27, 2013, New York, New York, USA.

responses to the membership questions. The key challenge we address in this paper is how to design a learning algorithm that runs in polynomial time, asks as few questions as possible and exactly identifies the intended query.

We prove the following:

1. Learning quantified Boolean queries is intractable: A doubly exponential number of questions is required (§2). Within a special class of quantified Boolean queries known as *qhorn* (§2.1), we prove two subclasses are exactly and efficiently learnable: *qhorn-1* (§2.1.3) and its superset *role-preserving qhorn* (§2.1.4) with constant limits on *causal density* (Def. 2.6).

2. We design an optimal algorithm to learn qhorn-1 queries using $O(n \lg n)$ questions where n is the number of propositions in a query (§3.1).

3. We design an efficient algorithm to learn role-preserving qhorn queries using $O(kn \lg n + n^{\theta+1})$ questions where k is *query size* (Def. 2.5), and θ is causal density (§3.2).

We also formalize a query verification model where the user specifies an entire query within the role-preserving qhorn query class. A verification algorithm then asks the user a set of membership questions known as the *verification set*. Each query has a unique verification set. The verification algorithm classifies some questions in the set as answers and others as non-answers. The query is incorrect if the user disagrees with any of the query's classification of questions in the verification set. We design a verification algorithm that asks $O(k)$ membership questions (§4).

2. PRELIMINARIES

Before we describe our query learning and verification algorithms, we first describe our data model — *nested relations* — and the qhorn query class.

DEFINITION 2.1. *Given the sets $D_1, D_2, ..., D_m$, $\mathcal{R}$ is a **relation** on these m sets if it is a set of m-tuples $(d_1, d_2, ..., d_m)$ such that $d_i \in D_i$ for $i = 1, ..., m$. $D_1, ..., D_m$ are the the domains of $\mathcal{R}$.*

DEFINITION 2.2. *A **nested relation** $\mathcal{R}$ has at least one domain D_i that is a set of subsets (powerset) of another relation $\mathcal{R}_i$. This $\mathcal{R}_i$ is said to be an embedded relation of $\mathcal{R}$.*

DEFINITION 2.3. *A relation $\mathcal{R}$ is a **flat relation** if all its domains $D_1, ..., D_m$ are not powersets of another relation.*

For example, a flat relation of chocolates can have the following schema:

```
Chocolate(isDark, hasFilling, isSugarFree,
hasNuts, origin)
```

A nested relation of boxes of chocolates can have the following schema:

```
Box(name, Chocolate(isDark, hasFilling,
isSugarFree, hasNuts, origin))
```

In this paper, we analyze queries over a nested relation with single-level nesting, i.e. the embedded relation is flat. The `Box` relation satisfies single-level nesting as the `Chocolate` relation embedded in it is flat. To avoid confusion, we refer to elements of the nested relation as *objects* and elements of the embedded flat relation as *tuples*. So the boxes are objects and the individual chocolates are tuples.

DEFINITION 2.4. *A **Boolean query** maps objects into either* answers *or* non-answers.

The atoms of a query are Boolean propositions such as:

$$p_1 : c.\texttt{isDark},\ p_2 : c.\texttt{hasFilling},$$
$$p_3 : c.\texttt{origin = Madagascar}$$

A complete query statement assigns quantifiers to expressions on propositions over attributes of the embedded relation. For example:

$$\forall c \in \texttt{Box.Chocolates}\ (p_1) \wedge \\ \exists c \in \texttt{Box.Chocolates}\ (p_2 \wedge p_3) \tag{1}$$

A box of chocolates is an answer to this query if every chocolate in the box is dark and there is at least one chocolate in the box that has filling and comes from Madagascar.

Given a collection of propositions, we can construct an abstract Boolean representation for the tuples of the nested relation. For example, given propositions p_1, p_2, p_3, we can transform the chocolates from the data domain to the Boolean domain as seen in Figure 1.

Box	origin	isSugar Free	is Dark	has Filling	has Nuts
Global Ground	Madagascar	1	1	1	0
	Belgium	1	0	0	1
	Germany	1	1	1	1
Europe's Finest	Belgium	1	1	0	0
	Belgium	0	1	0	1
	Sweden	0	1	1	1

Box (S)	p_1:is Dark	p_2: has Filling	p_3: origin = Madagascar
	x_1	x_2	x_3
S_1	1	1	1
	0	0	0
	1	1	0
S_2	1	0	0
	1	1	0

Figure 1: Transforming data from its domain into a Boolean domain.

Thus, each proposition p_i is replaced with a Boolean variable x_i. We rewrite the Boolean query (1) as follows:

$$\forall t \in S\ (x_1) \wedge \\ \exists t \in S\ (x_2 \wedge x_3)$$

where S is the set of Boolean tuples for an object. This Boolean representation allows us to create learning and verification algorithms independent of the data domain or of the actual propositions that the user writes.

To support this Boolean representation of tuples, however, we assume that (i) it is relatively efficient to construct an actual data tuple from a Boolean tuple and that (ii) the true/false assignment to one proposition does not interfere with the true/false assignments to other propositions. The propositions $p_m : c.\texttt{origin = Madagascar}$ and $p_b : c.\texttt{origin = Belgium}$ interfere with each other as a chocolate cannot be both from Madagascar and Belgium: $p_m \rightarrow \neg p_b$ and $p_b \rightarrow \neg p_m$.

With three propositions, we can construct 2^3 possible Boolean tuples, corresponding to the 2^3 possible true or false assignments to the individual propositions, i.e. we can construct 8 different chocolate classes. With n propositions, we can construct 2^n Boolean tuples.

There are 2^{2^n} possible sets of Boolean tuples or unique objects. With our three chocolate propositions, we can construct 256 boxes of distinct mixes of the 8 chocolate classes.

Since, a Boolean query maps each possible object into an answer or a non-answer, it follows that there are $2^{2^{2^n}}$ distinguishable Boolean queries (for $n = 3$, about 10^{77}). If our goal is to learn *any* query from n simple propositions by asking users to label objects as answers or non-answers, i.e. asking *membership questions*, then we would have to distinguish between $2^{2^{2^n}}$ queries using $\Omega(\lg(2^{2^{2^n}}))$ or 2^{2^n} questions.

Since this ambitious goal of learning any query with few membership questions is doomed to fail, we have to constrain the query space. We study the learnability of a special space of queries, which we refer to as *qhorn*.

2.1 Qhorn

Qhorn has the following properties:

1. It supports if-then query semantics via quantified *Horn* expressions: $\forall t \in S\ (x_1 \wedge x_2 \rightarrow x_3)$. A Horn expression has a conjunction of body variables that imply a single head variable. The degenerate *headless* Horn expression is simply a quantified conjunction of body variables ($\exists t \in S(x_1 \wedge x_2)$) and the degenerate *bodyless* Horn expression is simply a single quantified variable ($\forall t \in S(\mathbf{T} \rightarrow x_1) \equiv \forall t \in S(x_1)$).

2. It requires at least one positive instance for each Horn expression via a *guarantee clause*. Thus, we add the existential clause $\exists t \in S\ (x_1 \wedge x_2 \wedge x_3)$ to the expression $\forall t \in S\ (x_1 \wedge x_2 \rightarrow x_3)$ to get a complete query. Note that the expression $\exists t \in S\ (x_1 \wedge x_2 \rightarrow x_3)$ is implied by its guarantee clause $\exists t \in S\ (x_1 \wedge x_2 \wedge x_3)$.

   We justify the naturalness of guarantee clauses with the following example: consider a user looking for a box of only sugar-free chocolates. Without the guarantee clause, an empty box satisfies the user's query. While such a result is logical, we contend that most users would not consider the result as representative of sugar-free chocolate boxes.

3. It represents queries in a normalized form: *conjunctions of quantified (Horn) expressions.*

We use a shorthand notation for queries in qhorn. We drop the implicit '$t \in S$', the '$\wedge$' symbol and the guarantee clause. Thus, we write the query

$$\forall t \in S\ (x_1 \wedge x_2 \rightarrow x_3)\ \wedge\ \exists t \in S\ (x_1 \wedge x_2 \wedge x_3) \wedge \forall t \in S\ (x_4)\ \wedge\ \exists t \in S\ (x_4)\ \wedge\ \exists t \in S\ (x_5)$$

as $\forall x_1 x_2 \rightarrow x_3\ \forall x_4\ \exists x_5$.

2.1.1 Qhorn's Equivalence Rules

R1 The query representation $\exists x_1 x_2 x_3\ \exists x_1 x_2\ \exists x_2 x_3$ is equivalent to $\exists x_1 x_2 x_3$. This is because if a set contains a tuple that satisfies $\exists x_1 x_2 x_3$, that tuple will also satisfy $\exists x_1 x_2$ and $\exists x_2 x_3$. An existential conjunction over a set of variables **dominates** any conjunction over a subset of those variables.

R2 The query representation $\forall x_1 x_2 x_3 \rightarrow h\ \forall x_1 x_2 \rightarrow h\ \forall x_1 \rightarrow h$ is equivalent to $\forall x_1 \rightarrow h\ \exists x_1 x_2 x_3 \rightarrow h$. This is because h has to be true whenever x_1 is true regardless of the true/false assignment of x_2, x_3. Thus a universal Horn expression with body variables B and head variable h **dominates** any universal Horn expression with body variables B' and head variable h where $B' \supseteq B$.

R3 The query representation $\forall x_1 \rightarrow h\ \exists x_1 x_3$ is equivalent to $\forall x_1 \rightarrow h\ \exists x_1 x_3 h$. Again, this equivalence is because h has to be true whenever x_1 is true.

2.1.2 Learning with Membership Questions

A **membership question** is simply an object along with its nested data tuples. The user responds to such a question by classifying the object as an answer or a non-answer for their intended query.

Given a collection of n propositions on the nested relation, the learning algorithm constructs a membership question in the Boolean domain: a set of Boolean tuples on n Boolean variables $x_1, ..., x_n$ — a variable for each proposition. Such a set is transformed into an object in the data domain before presentation to the user.

For brevity, we describe a membership question in the Boolean domain only. As a notational shorthand, we use 1^n to denote a Boolean tuple where all variables are true. We use lowercase letters for variables and uppercase letters for sets of variables.

The following definitions describe two structural properties of qhorn queries that influence its learnability:

DEFINITION 2.5. ***Query size**, k, is the number of expressions in the query.*

DEFINITION 2.6. ***Causal Density**, θ, is the maximum number of distinct non-dominated universal Horn expressions for a given head variable h.*

Conceptually, universal Horn expressions represent causation: whenever the body variables are true, the head variable has to be true. If a head variable has many universal Horn expressions, it has many causes for it to be true and thus has a high causal density.

The following inequality between causal density, θ and query size k holds: $0 \leq \theta \leq k$. We would expect users' queries to be small in size $k = O(n)$ and to have low causal density θ.

A query class is *efficiently* learnable if (i) the number of membership questions that a learning algorithm asks the user is polynomial in the number of propositions n and query size k and (ii) the learning algorithm runs in time polynomial in n and k. Question generation needs to be in polynomial time to ensure interactive performance. This requirement entails that the number of Boolean tuples per question is polynomial in n and k. A query class is *exactly* learnable if we can learn the exact target query that satisfies the user's responses to the membership questions.

Due to the following theorem, qhorn cannot be efficiently and exactly learned with a tractable number of questions (even when query size is polynomially bounded in the number of propositions ($k = n$) and causal density has an upper bound of one ($\theta = 1$)).

THEOREM 2.1. *Learning qhorn queries where variables can repeat $r \geq 2$ times requires $\Omega(2^n)$ questions.*

Proof: See full version of this paper [1].

Qhorn's intractability does not mean that we cannot construct efficiently and exactly learnable qhorn subclasses. We describe two such sub-classes:

2.1.3 Qhorn-1

Qhorn-1 defines certain syntactic restrictions on qhorn. Not counting guarantee clauses, if a query has k distinct expressions ($1 \leq k \leq n$) and each expression i has body variables B_i and a head variable h_i, such that $B = B_1 \cup ... \cup B_k$ is the collection of all body variables and $H = \{h_1, ...h_m\}$ is the set of all head variables then the following restrictions hold in qhorn-1:

1. $B_i \cap B_j = \emptyset \vee B_i = B_j$ if $i \neq j$
2. $h_i \neq h_j$ if $i \neq j$
3. $B \cap H = \emptyset$

The first restriction ensures that different head variables can either share the exact same set of body variables or have disjoint bodies. The second restriction ensures that a head variable has only one body. Finally, the third restriction ensures that a head variable does not reappear as a body variable. Effectively, qhorn-1 has *no variable repetition*: a variable can appear **once** either in a set of body variables or as a head variable. The following diagram labels the different components of a qhorn-1 query.

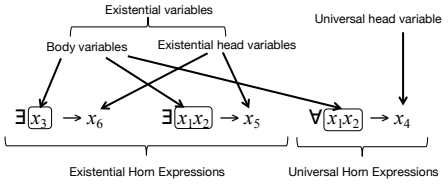


Figure 2: The different components of a qhorn-1 query.

Note that qhorn-1 queries have a maximum query size k of n and have a causal density θ of at most one. From an information-theoretic perspective, $\Omega(n \lg n)$ membership questions are required to learn a target query in qhorn-1 [1].

2.1.4 Role-preserving qhorn

In role-preserving qhorn queries, variables can repeat many times, but across universal Horn expressions head variables can only repeat as head variables and body variables can only repeat as body variables. For example, the following query is in role-preserving qhorn

$$\forall x_1x_4 \rightarrow x_5 \ \forall x_3x_4 \rightarrow x_5 \ \forall x_2x_4 \rightarrow x_6 \ \exists x_1x_2x_3 \ \exists x_1x_2x_5x_6$$

while the following query is not in role-preserving qhorn

$$\forall x_1x_4 \rightarrow x_5 \ \forall x_2x_3x_5 \rightarrow x_6$$

because x_5 appears both as a head variable and a body variable in two universally quantified Horn expressions. Existential Horn expressions in role-preserving qhorn are rewritten as existential conjunctions and variables do not have *roles* in these conjunctions. Thus, existential conjunctions can contain one or more head variables (e.g. $\exists x_1x_2x_5x_6$ in the first query). The following diagram labels the different components of a role-preserving qhorn query.

Both query size and causal density play a role in the behavior of learning and verification algorithms. Once we remove the syntactic restriction of variables appearing at most once, the size of a target query instance is no longer polynomially bounded in n. Thus, the complexity of learning and verification algorithms for role-preserving qhorn queries is parameterized by k, θ and n. We would expect user queries

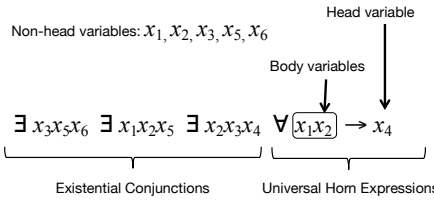


Figure 3: The different components of a role-preserving qhorn query.

to have low causal densities and to be small in size. Provided that θ has a constant upper bound, then we can efficiently learn role-preserving queries.

3. QUERY LEARNING

3.1 Learning qhorn-1

THEOREM 3.1. *$O(n \lg n)$ questions are sufficient to learn qhorn-1 queries in polynomial time.*

Proof: The learning algorithm breaks down query learning into a series of small tasks. First, it classifies all variables into either *universal head* variables or *existential* variables (Fig. 2 describes qhorn-1 terminology). Second, it learns the body variables (if any) for each universal head variable. Finally, it learns existential Horn expressions. We show that each task requires at most $O(n \lg n)$ membership questions (Section 3.1.1, Lemmas 3.2 and 3.3), thus proving that the learning algorithm asks $O(n \lg n)$ questions. □

3.1.1 Learning universal head variables

The simplest learning task is to determine whether a variable is a universal head variable. Suppose we have three variables: x_1, x_2, x_3. To determine if x_1 is the head of a universal Horn expression, we ask the user if the set $\{111, 011\}$ is an answer. By setting the other variables (x_2, x_3) to be always true, we are setting all potential body variables of x_1 to true. We are also neutralizing the effect of other unknown head variables on the outcome of a membership question. If the set $\{111, 011\}$ is an answer, then we are sure that x_1 is not a universal head variable because it can exist with a false value as long as at least one tuple has a true value for it. If the set is a non-answer, then we learn that x_1 is a universal head variable.

We need one question to determine whether a variable is a universal head variable and we need $O(n)$ time to generate each question — the time to construct a set with two tuples of size n. Thus, we learn which variables are universal head variables, U, and which variables are existential variables, E, in polynomial time.

3.1.2 Learning body variables of universal Horn expressions

DEFINITION 3.1. *Given a universal head variable h and a subset of existential variables $V \subseteq E$, a **universal dependence question** on h and V is a membership question with two tuples: 1^n and a tuple where h and V are false and all other variables are true.*

If a universal dependence question on h and V is an answer, then we learn that a subset of h's body variables is in V. This is because when the conjunction of body variables is not satisfied, the head variable can be false. We say that

h *depends* on some variables in V. If the question is a non-answer, then we learn that h's body variables are a subset of $E - V$; h has no body variables in V because in qhorn-1, h can have at most one body.

The most straightforward way to learn the body variables, B, of one universal variable is with $O(|E|) = O(n)$ universal dependence questions: we serially test if h depends on each variable $e \in E$. This means we use $O(n^2)$ questions to determine the body variables for all universal variables. We can do better.

We perform a binary search for h's body variables in E. If h has B body variables, we ask $O(|B| \lg n)$ instead of $O(n)$ questions to determine B. Suppose we have four variables x_1, x_2, x_3, x_4 such that x_1 is a universal head variable and all other variables are existential variables. x_2, x_3, x_4 are potential body variables for x_1. If the set $\{1^n, 0^n\}$ is a non-answer then x_1 is independent of all other variables and it has no body. If the set is an answer, we divide and conquer the variables. We ask if x_1 universally depends on half the variables, $\{x_2, x_3\}$, with the set $\{1^n, 0001\}$. If the set is a non-answer then we eliminate half the variables, $\{x_2, x_3\}$, from further consideration as body variables. We know that a body variable has to exist in the remaining half and since, x_4 is the last remaining variable, we learn the expression $\forall x_4 \rightarrow x_1$. If the set $\{1^n, 0001\}$ is an answer, then we know at least one body variable exists in $\{x_2, x_3\}$ and we continue the search for body variables in $\{x_2, x_3\}$, making sure that we also search the other half $\{x_4\}$ for body variables.

LEMMA 3.2. *$O(n \lg n)$ universal dependence questions are sufficient to learn the body variables of all universal head variables.*

Proof: Suppose we partition all variables into m non-overlapping parts of sizes $k_1, k_2, ..., k_m$ such that $\sum_{i=1}^{m} k_i = n$. Each part has at least one body variable and at least one universal head variable. Such a query class is in qhorn-1 as all body variables are disjoint across parts and head variables cannot reappear as head variables for other bodies or in the bodies of other head variables.

Given a head variable h_i, we can determine its body variables B_i using the binary search strategy above: we ask $O(|B_i| \lg n)$ questions (it takes $O(\lg n)$ questions to determine one body variable). For each additional head variable, h_i', that shares B_i, we require at most $1 \lg n$ questions: we only need to determine that h_i' has one body variable in the set B_i. Thus to determine all variables and their roles in a part of size k_i with $|B_i|$ body variables and $|H_i|$ head variables we need $O(|B_i| \lg n + |H_i| \times 1 \lg n) = O(k_i \lg n)$ questions. Since there are m parts, we ask a total of $O(\sum_{i=1}^{m} k_i \lg n) = O(n \lg n)$ questions. □

Since universal dependence questions consist of two tuples we only need $O(n)$ time to generate each question. Thus, the overall running time of this subtask is in polynomial time.

3.1.3 Learning existential Horn expressions

After learning universal Horn expressions, we have established some non-overlapping distinct bodies and their universal head variables. Each variable in the remaining set of existential variables, can either be (i) an existential head variable of one of the existing bodies or (ii) an existential head variable of a new body or (i) a body variable in the new body. We use *existential independence* questions to differentiate between these cases.

DEFINITION 3.2. *Given two disjoint subsets of existential variables $X \subset E, Y \subset E, X \cap Y = \emptyset$, an **existential independence question** is a membership question with two tuples: (i) a tuple where all variables $x \in X$ are false and all other variables are true and (ii) a tuple where all variables $y \in Y$ are false and all other variables are true.*

If an independence question between two existential variables x and y is an answer then either:

1. x and y are existential head variables of the same body
2. or x and y are not in the same Horn expression.

We say that x and y are *independent* of each other. Two sets X and Y are independent of each other if all variables $x \in X$ are independent of all variables $y \in Y$. Conversely, if an independence question between x and y is a non-answer then either:

1. x and y are body variables in the same body or
2. y is an existential head variable and x is in its body or
3. x is an existential head variable and y is in its body

We say that x and y *depend* on each other. If sets X and Y depend on each other then at least one variable $x \in X$ depends on one variable $y \in Y$.

Given an existential variable e, if we discover that e depends on a body variable b of a known set of body variables B, then we learn that e is an existential head variable in the Horn expression: $\exists B \rightarrow e$.

Otherwise, we find all existential variables D that e depends on. We can find all such variables with $O(|D| \lg n)$ existential independence questions using the binary search strategy of Section 3.1.2.

Knowing that D depends on e only tell us that one of the following holds: (i) A subset H of D are existential head variables for the body of $e \cup (D - H)$ or (ii) e is a head variable and D is a body. To differentiate between the two possibilities we make use of the following rule: *If two variables x, y depend on z but x and y are independent then z is a body variable and x, y are head variables.* If we find a pair of independent variables h_1, h_2 in D, we learn that x must be a body variable. If we do not find a pair of independent variables in D then we may assume that x is an existential head variable and all variables in D are body variables.

After finding head variables in D, we can determine the roles of the remaining variables in D with $|D| = O(n)$ independence questions between h_1 and each variable $d \in D - h_1$. If h_1 and d are independent then d is an existential head variable, otherwise d is a body variable.

Our goal, therefore, is to locate a definitive existential head variable in D by searching for an independent pair of variables.

DEFINITION 3.3. *An **independence matrix question** on D variables consists of $|D|$ tuples. For each variable $d \in D$, there is one tuple in the question where d is false and all other variables are true.*

Suppose we have four variables $x_1, ..., x_4$; $D = \{x_2, x_3, x_4\}$ and D depends on x_1. $\{\mathbf{1011}, \mathbf{1101}, \mathbf{1110}\}$ is a matrix question on D. If such a question is an answer then there is *at least a pair* of head variables in D: the question will always contain a pair of tuples that ensure that each head and the body is true. For example if x_2, x_4 are head variables then tuples $\{\mathbf{1011}, \mathbf{1110}\}$ in the question satisfy the Horn expressions: $\exists x_1 x_3 \rightarrow x_2, \exists x_1 x_3 \rightarrow x_4$. If *at most one* vari-

able in D is a head variable, then there is no tuple in the matrix question where all body variables are true and the head variable is true and the question is a non-answer. For example, if only x_4 is a head variable, then the tuple, 1111 that satisfies the Horn expression $\exists x_1x_2x_3 \rightarrow x_4$ is absent from the question.

LEMMA 3.3. *Given an existential variable x and its dependents D, we can find an existential head variable in D with $O(|D| \lg |D|)$ independence matrix questions of $O(|D|)$ tuples each if at least two head variables exist in D.*

Algorithm 1 Get Head

```
x: an existential variable
D: the dependents of x, |D| ≥ 1
D1 ← D, D2 ← ∅, D3 ← ∅
while D1 ≠ ∅ do
    if isAnswer(Ask(MatrixQuestion(x, D1))) then
        if |D1| = 2 ∧ D2 = ∅ then return D1
        else if |D1| > 2 ∧ D2 = ∅ then
            Split D1 into D1 (1st half) and D3 (2nd half)
        else if |D2| = 1 then return D2
        else
            Split D2 into D2 (1st half) and D3 (2nd half)
            D1 ← D1 − D3
        end if
    else
        if D3 = ∅ then return ∅
        else if |D3| = 1 then return D3
        else
            Split D3 into D2 (1st half) and D3 (2nd half)
            D1 ← D1 ∪ D2
        end if
    end if
end while
```

Proof. Consider the 'GetHead' procedure in Alg. 1 that finds an existential head variable in the set D of dependents of variable x. The central idea behind the 'GetHead' procedure is if the user responds that a matrix question on D_1 ($D_1 \subseteq D$) is an answer, then a pair of head variables must exist in D_1 and we can eliminate the remaining variables $D-D_1$ from further consideration. Otherwise, we know that at most one head variable exists in D_1 and another exists in $D - D_1$ so we can eliminate D_1 from further consideration and focus on finding the head variable in $D - D_1$.

Each membership question eliminates half the variables from further consideration as head variables. Thus, we require only $O(\lg |D|) = O(\lg n)$ questions to pinpoint one head variable.

Then, we ask $O(|D|)$ questions to differentiate head from body variables in D. If we do not find head variables in $|D|$ then we may assume that x is a head variable and all variables in D are body variables. Once we learn one existential Horn expression, we process the remaining existential variables in E. If a variable depends on any one of the body variables, B, of a learned existential Horn expression, it is a head variable to all body variables in B.

Suppose a query has m distinct existential expressions with $k_1, ..., k_m$ variables each, then $\sum_{i=1}^{m} k_i < n$. The size of each set of dependent variables for each expression i is $k_i - 1$. So the total number of questions we ask is $\sum_{i=1}^{m} \big(O(k_i \lg n) + O(\lg k_i) + O(k_i)\big) = O(n \lg n)$

Note, however, that each matrix question has $O(|D|) = O(n)$ tuples of n variables each and therefore requires $O(n^2)$ time to generate. If we limit the number of tuples per question to a constant number, then we increase the number of questions asked to $\Omega(n^2)$ [1].

3.2 Learning role-preserving qhorn

Since some queries are more complex than others within the role-preserving qhorn query class it is natural to allow our learning algorithm more time, more questions and more tuples per question to learn the more complex target queries. One can argue that such a powerful learning algorithm may not be practical or usable as it may ask many questions with many tuples each. If we assume that user queries tend to be simple (i.e they are small in size k (Def. 2.5) and have low causal densities θ (Def. 2.6)), then such an algorithm can be effective in the general case.

Role-preserving qhorn queries contain two types of expressions: universal Horn expressions ($\forall x_1x_2... \rightarrow h$) and existential conjunctions ($\exists x_1x_2...$) (Fig. 3 describes role-preserving qhorn terminology). In this section, we show that we can learn all universal Horn expressions with $O(n^{\theta+1})$ questions and all existential conjunctions with $O(kn \lg n)$ questions[1]. Since run-time is polynomial in the number of questions asked, our run-time is $poly(nk)$ and $poly(n^{\theta})$ respectively. By setting a constant upper limit on the causal density of a head variable we can learn role-preserving qhorn queries in $poly(nk)$ time.

We employ a Boolean lattice on the n variables of a query to learn the query's expressions. Fig. 4 illustrates the Boolean lattice and its key properties. Each point in the lattice is a tuple of true or false assignments to the variables. A lattice has $n + 1$ levels. Each level l starting from level 0 consists of tuples where exactly l variables are false. A tuple's children are generated by setting exactly one of the true variables to false. Tuples at l have out-degree of $n - l$, i.e. they have $n - l$ children and in-degree of l or l parents. A tuple has an *upset* and a *downset*. These are visually illustrated in Fig. 4. If a tuple is not in the upset or downset of another tuple, then these two tuples are *incomparable*.

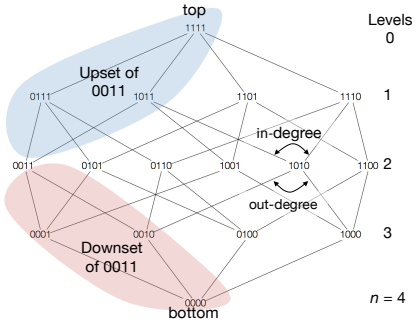


Figure 4: The Boolean lattice on four variables.

The gist of our lattice-based learning algorithms is as follows:

1. We map each tuple in the lattice to a distinct expression. This mapping respects a certain *generality* ordering of expressions. For example, the lattice we use to learn existential conjunctions maps the top tuple in the lattice

[1]We show lower bounds of $\Omega(\frac{n}{\theta}^{\theta-1})$ for learning universal Horn expressions and $\Omega(nk)$ for learning existential conjunctions in [1]

to the most specific conjunction $\exists x_1 x_2 ... x_n$; tuples in the level above the bottom of the lattice map to the more general conjunctions $\exists x_1$, $\exists x_2$, ..., $\exists x_n$ (§3.2.2). The exact details of this mapping for learning universal Horn expressions and learning existential conjunctions are described in the following section.

2. We search the lattice in a *top-to-bottom* fashion for the tuple that *distinguishes* or maps to the target query expression. The learning algorithm generates membership questions from the tuples of the lattice and the user's responses to these questions either *prune* the lattice or *guide* the search.

3.2.1 Learning universal Horn expressions

We first determine head variables of universal Horn expressions. We use the same algorithm of (§3.1.1). The algorithm uses $O(n)$ questions. We then determine *bodyless* head variables. To determine if h is bodyless, we construct a question with two tuples: 1^n and a tuple where h and all existential variables are false and all other variables are true. If the question is a non-answer then h is bodyless. If h is not bodyless then we utilize a special lattice (Fig. 5) to learn h's different bodies. In this lattice, we neutralize the effect of other head variables by fixing their value to true and we fix the value of h to false.

DEFINITION 3.4. *A universal Horn expression for a given head variable h is **distinguished** by a tuple if the true variables of the tuple represent a complete body for h.*

Thus, each tuple in the lattice distinguishes a unique universal Horn expression. For example, consider the target query:

$$\forall x_1 x_4 \rightarrow x_5 \ \forall x_3 x_4 \rightarrow x_5 \ \forall x_1 x_2 \rightarrow x_6$$
$$\exists x_1 x_2 x_3 \ \exists x_2 x_3 x_4 \ \exists x_1 x_2 x_5 \ \exists x_2 x_3 x_5 x_6$$

In the target query, the head variable x_5 has two universal Horn expressions:

$$\forall x_1 x_4 \rightarrow x_5 \ \forall x_3 x_4 \rightarrow x_5$$

In Fig. 5, we marked the two tuples that *distinguish* x_5's universal Horn expressions: 100101 and 001101. Notice that the universal Horn expressions are ordered from most to least specific. For example the top tuple of the lattice in Fig. 5 is the distinguishing tuple for the expression $\forall x_1 x_2 x_3 x_4 \rightarrow x_5$. While the bottom tuple is the distinguishing tuple for the expression $\forall x_5$. Our learning algorithm searches for distinguishing tuples of only *dominant* universal Horn expressions.

A membership question with a distinguishing tuple and the all-true tuple (a tuple where all variables are true) is a non-answer for one reason only: *it violates the universal Horn expression it distinguishes.* This is because the all-true tuple satisfies all the other expressions in the target query and the distinguishing tuple sets a complete set of body variables to true but the head to false. More importantly, all such membership questions constructed from tuples in the upset of the distinguishing tuple are non-answers and all questions constructed from tuples in the downset of the distinguishing tuple are answers. Thus, the key idea behind the learning algorithm is to efficiently search the lattice to find a tuple where questions constructed from tuples in the upset are non-answers and questions constructed from tuples in the downset are answers.

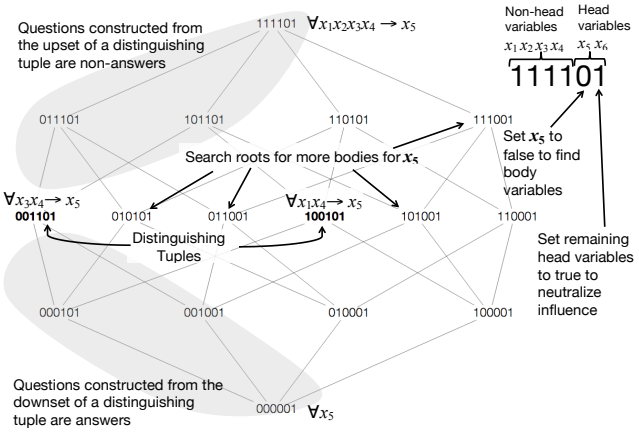


Figure 5: Learning bodies for a given head variable

Given a head variable h and n (non-head) variables, we use a Boolean lattice on n variables (with $h = 0$ and all other head variables set to true). We construct a membership question with a tuple t from the lattice and the all-true tuple — a tuple where all variables, including the head variable, are true. We begin by describing how we can use the lattice to find just one set of body variables that determine h with $O(n)$ questions. We start at the top of the lattice, we construct a question from the top tuple and proceed as follows:

1. If the question is an answer, then it does not contain an entire set of body variables that determine h. We prune its *downset*. We move to the next tuple on the same level of the lattice.

2. If the question is a non-answer then some of the true variables in t form a body and we move down the lattice (skipping previously pruned tuples). If all of t's children are answers, then t is a *universal distinguishing tuple* for the head variable h.

Once we find a body, we can safely eliminate its *upset*. Any body in the upset is *dominated* (Rule 2) by the discovered distinguishing tuple. Looking at Fig. 5, we notice that the upset simply contains all tuples where all body variables of the distinguishing tuple are true. The remaining lattice structure is rooted at tuples where one of the body variables is false. Since two incomparable bodies need to differ on at least one body variable, we set one body variable to false and search the resulting sub-lattices for bodies.

THEOREM 3.4. *$O(n^\theta)$ membership questions, where θ is the causal density of the given head variable h, are sufficient to the learn the θ universal Horn expressions of h.*

Proof: Let b_i denote the number of body variables for each distinguishing tuple t_i found. Initially we set b_0 to n and we search the entire lattice or the n sub-lattices rooted at the tuples where exactly one Boolean variable is false. In Fig. 5 those are the tuples at level 1: {0111*01*, 1011*01*, 1101*01*, 1110*01*}.

If the first distinguishing tuple found has $|B_1|$ true variables, then we need to search $|B_1|$ sub-lattices for bodies. For example, after finding the distinguishing tuple 0011*01*, we continue searching for more distinguishing tuples from $|B_1| = 2$ roots: {1101*01*, 1110*01*}.

Suppose we find a second distinguishing tuple: 1001*01* with B_2 body variables; then we need to search for

more bodies in the sub-lattices rooted at tuples where one of each body variable from the distinct bodies are set to false. Our new $|B_1| \times |B_2|$ roots are: {010*101*, 011*001*, 101*001*, 111*001*}. These *search roots* are illustrated in Fig. 5.

In the worst case, we ask $O(n)$ questions to find a body. Thus to determine all θ expressions for a universal head variable, an upper bound on the number of questions, Q, is:

$$Q \leq \begin{array}{l}(n) + (|B_1| + n) + (|B_1| \times |B_2| + n) + ...+ \\ (|B_1| \times |B_2| \times ... \times |B_\theta|)\end{array}$$

$$Q \leq n\theta + \sum_{b=1}^{\theta}(\prod_{i=1}^{b}|B_i|) \leq n\theta + \sum_{i=1}^{\theta}(n^i) = O(n^\theta) \quad \square$$

Since there are $O(n)$ head variables and for each head variable we ask $O(n^\theta)$ questions to determine its universal Horn expressions, we learn all universal Horn expression with $O(n \times n^\theta) = O(n^{\theta+1})$ questions.

3.2.2 Learning existential conjunctions

To learn existential conjunctions of a query we use the full Boolean lattice on all n variables of a query (including head variables).

DEFINITION 3.5. *An existential conjunction C is **distinguished** by a tuple if the true variables of the tuple are the variables of the conjunction.*

Thus, each tuple in the lattice distinguishes a unique existential conjunction. For example, consider the target query:

$$\forall x_1x_4 \rightarrow x_5\ \forall x_3x_4 \rightarrow x_5\ \forall x_1x_2 \rightarrow x_6$$
$$\exists x_1x_2x_3\ \exists x_2x_3x_4\ \exists x_1x_2x_5\ \exists x_2x_3x_5x_6$$

The conjunction $\exists x_2x_3x_5x_6$ is distinguished by the tuple 011011 in a six-variable Boolean lattice.

Existential conjunctions are ordered from most to least specific on the lattice. For example, the top tuple 111111 of a six-variable lattice is the distinguishing tuple for the expression $\exists x_1x_2x_3x_4x_5x_6$; the tuples {00001, 000010, 000100, 001000, 010000, 100000} at level five of the lattice are the distinguishing tuples for the expressions $\exists x_6$, $\exists x_5$, $\exists x_4$, $\exists x_3$, $\exists x_2$, $\exists x_1$ respectively.

Our learning algorithm searches for distinguishing tuples of a *normalized* target query. For example, the target query above is normalized to the following semantically equivalent query using (Rule 3):

$$\begin{array}{l}\forall x_1x_4 \rightarrow x_5\ \forall x_3x_4 \rightarrow x_5\ \forall x_1x_2 \rightarrow x_6 \\ \exists x_1x_2x_3x_6\ \exists x_2x_3x_4x_5\ \exists x_1x_2x_5x_6\ \exists x_2x_3x_5x_6\end{array} \quad (2)$$

This query has the following *dominant* conjunctions (which include guarantee clauses):

$$\exists x_1x_4x_5\ \exists x_1x_2x_3x_6\ \exists x_2x_3x_4x_5\ \exists x_1x_2x_5x_6\ \exists x_2x_3x_5x_6$$

A membership question with **all dominant** distinguishing tuples of a query is an answer: all existential conjunctions (including guarantee clauses) are satisfied. For example, a question with the tuples: {100110, 111001, 011110, 110011, 011011} is an answer for the target query above (2).

Replacing a distinguishing tuple with its children results in a non-answer: the existential conjunction of that tuple is no longer satisfied. For example replacing 011011 with its children {001011, 010011, 011001, 011010} results in a membership question where none of the tuples satisfy the expression $\exists x_2x_3x_5x_6$.

Replacing a distinguishing tuple with any tuple in its upset *that does not violate a universal Horn expression* still results in an answer.

Thus, the learning algorithm searches level-by-level from top-to-bottom for distinguishing tuples by detecting a change in the user's response to a membership question from answer to non-answer. The efficiency of the learning algorithm stems from *pruning*: when we replace a tuple with its children, we prune those down to a minimal set of tuples that still dominate all the distinguishing tuples.

We describe the learning algorithm (Alg. 2) with an example and then prove that the learning algorithm runs in $O(kn \lg n)$ time (Theorem. 3.5)[2].

Algorithm 2 Find Existential Distinguishing Tuples

```
T ← {1^n}                                   ▷ The top tuple.
D ← {}      ▷ D is the set of discovered distinguishing tuples.
while T ≠ ∅ do
    T' ← {}
    for t ∈ T do
        C ← Children(t)
        C ← RemoveUniversalHornViolations(C)
        T ← T − {t}
        if isAnswer(Ask(D ∪ T ∪ C ∪ T')) then
            T' ← T' ∪ Prune(C, T ∪ D)
        else
            D ← D ∪ {t}
        end if
    end for
    T ← T'
end while
return D
```

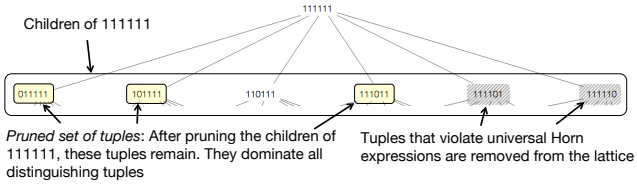
Algorithm 3 Prune

```
T: the tuples to prune
O: other tuples
K ← {}                                  ▷ K is the set of tuples to keep.
Split T into T1 (1st half) and T2 (2nd half).
while T1 ∪ T2 ≠ ∅ do
    if isAnswer(Ask(T1 ∪ K ∪ O)) then
        Split T1 into T1 (1st half) and T2 (2nd half).
    else
        if |T2| = 1 then
            K ← K ∪ T2
        else
            Add 1st half of T2 to T1. Set T2 to 2nd half of T2.
        end if
    end if
end while
return K
```

Suppose we wish to learn the existential conjunctions of the target query listed in (2). We use the six-variable Boolean lattice with the following modification: we remove all tuples that violate a universal Horn expression. These are tuples where the body variables of a universal Horn expression are true and the head variable is false. For example, the tuple 111110 violates $\forall x_1x_2 \rightarrow x_6$ is therefore removed from the lattice.

[2]In [1], we prove the algorithm's correctness and provide a lower bound of $O(nk)$ for learning existential conjunctions.

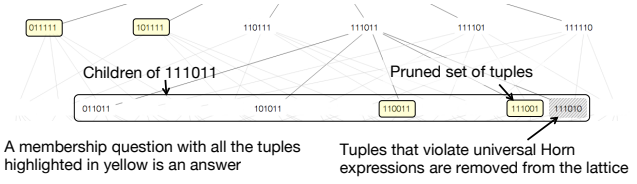
Level 1: We start at the top of the lattice. Since the tuple 111111 will satisfy any query, we skip to level one. We now construct a membership question with all the tuples of level 1 (after removing the tuples that violate universal Horn expressions: 111110, 111101): 111011, 110111, 101111, 011111. If such a question is a non-answer, then the distinguishing tuple is one level above and the target query has one dominant existential conjunction: $\exists x_1x_2x_3x_4x_5x_6$.



If the question is an answer, we need to search for tuples we can safely *prune*. So we remove one tuple from the question set and test its membership. Suppose we prune the tuple 110111, the question is still an answer since all conjunctions of the target query are still satisfied: the remaining set of tuples still dominate the distinguishing tuples of the target query.

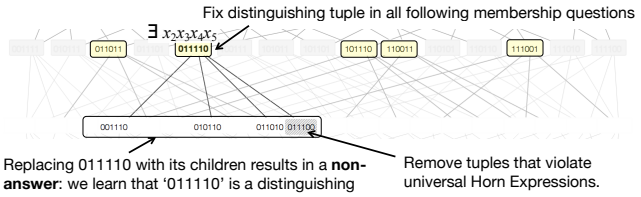
We then prune 011111. This question is a non-answer since no tuple satisfies the clause $\exists x_2x_3x_4x_5$. We put 011111 back in and continue searching at level one for tuples to prune. We are left with the tuples: 111011, 101111 and 011111. Note that we asked $O(n)$ questions to determine which tuples to safely prune. We can do better. In particular, we only need $O(\lg n)$ questions for each tuple we need to keep if we use a binary search strategy.

Level 2: We replace one of the tuples, 111011, with its children on level 2: $\{011011, 101011, 110011, 111001\}$. Note, that we removed 111010 because it violates $\forall x_1x_2 \rightarrow x_6$. As before we determine which tuples we can safely prune. We are left with $\{110011, 111001\}$.



Similarly we replace 101111 with its children on level 2: $\{001111, 100111, 101011, 101110\}$. We did not consider 101101 because it violates $\forall x_3x_4 \rightarrow x_5$. We can safely prune the children down to one tuple: 101110. We then replace 011111 with its children on level 2 and prune those down to $\{011011, 011110\}$. At the end of processing level 2, we are left with the tuples: $\{110011, 111001, 101110, 011011, 011110\}$. We repeat this process again now replacing each tuple, with tuples from level 3.

Level 3: When we replace 011110 with its children $\{010110, 011010, 001110\}$, we can no longer satisfy $\exists x_2x_3x_4x_5$. The question is a non-answer and we learn that 011110 is a distinguishing tuple and that $\exists x_2x_3x_4x_5$ is a conjunction in the target query. Note that we did not consider the child tuple 011100 because it violates the universal Horn expression $\forall x_3x_4 \rightarrow x_5$. We fix 011110 in all subsequent membership questions.



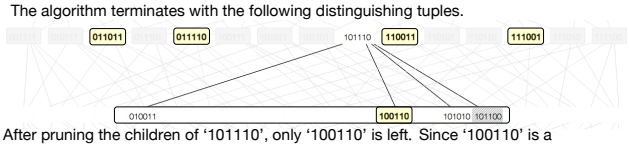
When we replace 011011 with its children $\{001011, 010011, 011001, 011010\}$, we can no longer satisfy $\exists x_2x_3x_5x_6$. The question is a non-answer and we learn that 011011 is a distinguishing tuple and that $\exists x_2x_3x_5x_6$ is a conjunction in the target query. We fix 011011 in all subsequent membership questions.

When we replace 111001 with its children $\{011001, 101001, 110001\}$, the question is a non-answer, and we learn that 111001 is distinguishing tuple and that $\exists x_1x_2x_3x_6$ is a conjunction in the target query. Note that we did not consider the tuple 111000 because it violates $\forall x_1x_2 \rightarrow x_6$. We fix 111001 in all subsequent membership questions.

We can replace 101110 with the children $\{001110, 100110, 101010\}$. Note that the child 101100 is removed because it violates $\forall x_1x_4 \rightarrow x_5$. We can safely prune the children down to one tuple 100110.

When we replace 110011 with its children $\{010011, 100011, 110001\}$, we can no longer satisfy $\exists x_1x_2x_5x_6$. Thus, the question is a non-answer and we learn that 110011 is a distinguishing tuple. Note that we did not consider the tuple 110010 because it violates $\forall x_1x_2 \rightarrow x_6$.

At this stage, we are left with the following tuples: $\{\mathbf{110011}, 100110, \mathbf{111001}, \mathbf{011011}, \mathbf{011110}\}$. At this point, we can continue searching for conjunctions in the downset of 100110 which is the distinguishing tuple for a known guarantee clause for the universal Horn expression: $\forall x_1x_4 \rightarrow x_5$. As an optimization to the algorithm, we do not search the downset because all tuples in the downset are dominated by 100110[3].



The learning algorithm terminates with the following distinguishing tuples $\{110011, 100110, 111001, 011011, 011110\}$ which represent the expressions:

$$\exists x_1x_2x_5x_6 \ \exists x_1x_4x_5 \ \exists x_1x_2x_3x_6 \ \exists x_2x_3x_5x_6 \ \exists x_2x_3x_4x_5$$

THEOREM 3.5. *The lattice-based learning algorithm asks $O(kn \lg n)$ membership questions where k is the number of existential conjunctions.*

Proof: Consider the cost of learning one distinguishing tuple t_l at level l. From the top of the Boolean lattice to t_l, there is at least one tuple t_i at each level i $(0 < i < l)$ that we did not prune and we traversed down from to get to t_l.

[3]We can relax the requirement of guarantee clauses for universal Horn expressions and our learning algorithms will still function correctly if they are allowed to ask about the membership of an empty set.

Let N_i be the set of t_i's siblings. At each level i, we asked at most $\lg|N_i|$ questions. $|N_i| = n-(i-1)$ or the out-degree of N_i's parent. In the worst-case, $l = n$, and the cost of learning t_l is $\sum_{i=1}^{n} \lg(n-(i-1)) \leq \sum_{i=1}^{n} \lg n = O(n \lg n)$. With k distinguishing tuples we ask at most $O(kn \lg n)$ questions. □

4. QUERY VERIFICATION

A query verifier constructs a set of membership questions to determine whether a given query is correct. The verifier will not find an alternate query if the query is incorrect. Thus, while query learning is a *search problem* — a learner searches for the one correct query that satisfies the user's responses to membership questions; query verification is the *decision problem* — a verifier decides if a given query is correct or incorrect given the user's responses to membership questions.

Our approach to query verification is straightforward: for a given role-preserving qhorn[4] query q_g, we generate a *verification set* of $O(k)$ membership questions, where k is the number of expressions in q_g. Note that our learning algorithm for role-preserving qhorn queries asks $O(n^{\theta+1} + kn \lg n)$ questions. If the user's intended query q_i is semantically different from the given query q_g, then for at least one of the membership questions M in the verification set $q_g(M) \neq q_i(M)$.

PROPOSITION 4.1. *A user's intended query q_i is semantically different from a given query q_g iff q_i and q_g have distinct sets of existential (Def. 3.5) and universal (Def. 3.4) distinguishing tuples.*

Suppose we try to learn the two role-preserving qhorn queries q_i and q_g. If q_i and q_g are semantically different, then our learning algorithm will terminate with distinct sets of existential(Def. 3.5) and universal(Def. 3.4) distinguishing tuples for each query. The verification set consists of membership questions that detect semantic differences between two queries by detecting differences in their respective sets of distinguishing tuples[5]. Fig. 6 lists six types of membership questions from which the verification algorithm constructs a verification set for a given query.

THEOREM 4.2. *A verification set with all membership questions of Fig. 6 surfaces semantic differences between the given query q_g and the intended query q_i by surfacing differences between the sets of distinguishing tuples of q_g and q_i.*

Proof: Case 1: q_i and q_g have different sets of dominant existential distinguishing tuples then by Lemma 4.3, questions A1 and N1 surface differences in the sets of dominant existential distinguishing tuples of q_g and q_i.

Case 2: q_i and q_g have different sets of dominant universal distinguishing tuples then

1. Both q_i and q_g classify h as a head variable. q_i has a dominant universal Horn expression $C_i : \forall B_i \rightarrow h$ (B is a set of body variables) and q_g has dominant universal Horn expressions of the form $\forall B_g \rightarrow h$.
   (a) If for *any* B_g in q_g, $B_i \subset B_g$ or $B_i \supset B_g$ then by Lemmas 4.4 and 4.5 questions A2 and N2 will surface this difference.
   (b) If for *all* B_g in q_g, B_i and B_g are incomparable then either (i) C_i's guarantee clause dominates q_g's existential expressions and q_g's set of existential distinguishing tuples does not have the distinguishing tuple for C_i's guarantee clause (See Case 1) or (ii) C_i's guarantee clause is dominated by an existential expression in q_g and by Lemma 4.6 question A3 surfaces the difference.
2. h is a head variable in q_i but is a non-head variable in q_g then by Lemma 4.7 question A4 surfaces the difference. □

LEMMA 4.3. *Let D_i be the set of q_i's dominant existential distinguishing tuples and let D_g be the set of q_g's dominant existential distinguishing tuples; membership questions A1 and N1 surface $D_i \neq D_g$.*

Proof: An existential distinguishing tuple represents an inflection point: all questions constructed with tuples in the distinguishing tuple's upset are answers and all questions constructed with only tuples in the rest of the lattice are non-answers. We use this feature to detect if $D_i \neq D_g$.

First, we define the following order relations over D_i and D_g:

1. $D_g \leq D_i$ if for every tuple $t_i \in D_i$, there exists a tuple $t_g \in D_g$ such that t_g is in the upset of t_i.
2. $D_g \geq D_i$ if all tuples in D_g are in the downset of D_i.
3. $D_g || D_i$, otherwise, i.e. they are incomparable.

Since $D_g \neq D_i$ only the following cases are possible:

Case 1: $D_g || D_i$ or $D_g > D_i$: D_g or membership question A1 is a non-answer to the user's intended query q_i. The user will detect the discrepancy as D_g is presented as an answer in q_g's verification set.

Case 2: $D_g < D_i$. Suppose all tuples in D_g are in the upset of one of D_i's tuples. Let $D_g(t)$ be the set of distinguishing tuples where we replace $t \in D_g$ with its children. There are $|D_g| = O(k)$ such sets. These sets form membership questions N1. For any $t \in D_g$, $D_g(t)$ is always a non-answer to q_g. However, for at least one tuple t, $D_g(t)$ is an answer to q_i. This is because if $D_g < D_i$ then at least one of D_i's tuples is a descendant of one of D_g's tuples, in which case $D_g(t)$ is still in the upset of that tuple and thus an answer. The user will detect the discrepancy as $D_g(t)$ is presented as a non-answer in q_g's verification set. □

Like existential distinguishing tuples, universal distinguishing tuples represent an inflection point. All tuples in the upset of the universal distinguishing tuple are non-answers (as all of h's body variables are true but h is false). All descendants of the universal distinguishing tuple are answers (as no complete set of h's body variables is true).

Let t_i be q_i's universal distinguishing tuple for an expression on the head variable h. Let t_g be one of q_g's universal distinguishing tuples for expressions on the head variable h. We define the following order relations between t_i and t_g:

1. $t_i \leq t_g$ if t_i is in the upset of t_g.
2. $t_i \geq t_g$ if t_i is in the downset of t_g.
3. $t_i || t_g$ if t_i and t_g are incomparable.

Consider two distinct (dominant) tuples t_{g_1} and t_{g_2} of the given query. By qhorn's equivalence rules(§2.1.1) queries t_{g_1}

[4]Since qhorn-1 is a sub-class of role-preserving qhorn, our verification approach works for both query classes.

[5]In [1], we describe how we normalize a given query and extract dominant distinguishing tuples from its expressions.

Answers				Non-Answers			
	Membership Questions	*# of Questions*	*Tuples / Question*		*Membership Questions*	*# of Questions*	*Tuples / Question*
A1	Distinguishing tuples for all dominant existential expressions (including guarantee clauses and existential Horn expressions)[1]	$O(1)$	$O(k)$	**N1**	For each distinguishing tuple in A1 that is not due to a guarantee clause: (i) Children of the distinguishing tuple[1] (ii) All other tuples from A1	$O(k)$	$O(n+k)$
A2	For each dominant universal Horn expression: (i) a tuple where all variables are true (ii) Children of the distinguishing tuple	$O(k)$	$O(n)$	**N2**	For each dominant universal Horn expression: (i) a tuple where all variables are true (ii) The distinguishing tuple	$O(k)$	$O(1)$
A3	For each dominant existential expression on C variables such that there are one or more universal Horn expressions $\forall B_1 \rightarrow h \ldots \forall B_\theta \rightarrow h$ where $B_i \subset C$ for $i = 1 \ldots \theta$: (i) a tuple where all variables are true (ii) Search roots: a tuple where one body variable from each body $B_1 \ldots B_\theta$ is false and all other variables in C are true and h is false	$O(k)$	$O(n^\theta)$				
A4	(i) A tuple where all variables are true (ii) A tuple for each non-head variable x such that x is false and all other variables are true	$O(1)$	$O(n)$				

[1]In constructing these questions, we do not violate universal Horn expressions: i.e. we set a head variable to true if the existential expression contains a body for the head variable

Figure 6: Membership questions of a verification set.

and t_{g_2} are incomparable ($t_{g_1} || t_{g_2}$). Consequently, for any two distinct tuples both $t_i < t_{g_1}$ and $t_i > t_{g_2}$ cannot hold.

LEMMA 4.4. *Membership question A2 detects $t_i > t_g$.*

Proof: Suppose, q_g has one universal distinguishing tuple t_g such that $t_i > t_g$. Then the membership question A2 that consists of the all-true tuple and t_g's children is an answer for q_g as none of t_g's children have all the body variables set to true, so the head variable can be false. If $t_i > t_g$ then q_i's universal Horn expression on h has a strict subset of the body variables represented by t_g. Therefore, in at least one of t_g's children, all of t_i's body variables are set to true and h is still false. Thus, A2 is a non-answer to q_i. For all other universal distinguishing tuples t_g of q_g, either $t_i > t_g$ or $t_i || t_g$. If $t_i || t_g$ then A2 is still an answer. □

LEMMA 4.5. *Membership question N2 detects $t_i < t_g$.*

Proof: Suppose, q_g has one universal distinguishing tuple t_g such that $t_i < t_g$. Then the membership question N2 that consists of the all-true tuple and t_g is a non-answer for q_g as t_g has all body variables set to true but the head variable h is false. If $t_i < t_g$ then q_i's universal Horn expression on h has a strict superset of the body variables represented by t_g. Therefore, t_g does not have all body variables set to true and h can be false. Thus, N2 is an answer to q_i.

For all other universal distinguishing tuples t_g of q_g, either $t_i < t_g$ or $t_i || t_g$. If $t_i || t_g$ then N2 is still a non-answer. □

LEMMA 4.6. *If*
- *h is a head variable in q_i and q_g.*
- *q_i has a dominant universal Horn expression $\forall M \rightarrow h$ which q_g does not have.*
- *q_g has universal Horn expressions $\forall B_1 \rightarrow h \ldots \forall B_\theta \rightarrow h$.*
- *$B_i || M$ for $i = 1 \ldots \theta$*
- *q_g has an existential expression on C variables ($\exists$ C) such that $C \supseteq M$ and $C \supset B_i$ for $i = 1 \ldots \theta$*

then A3 surfaces a missing universal Horn expression ($\forall M \rightarrow h$) from q_g.

Proof: Consider q_g's universal Horn expressions whose guarantee clauses are dominated by $\exists$ C:

$$\forall B_1 \rightarrow h, \forall B_2 \rightarrow h, \ldots \forall B_\theta \rightarrow h$$

such that $B_i \subset C$ for $i = 1 \ldots \theta$. To build A3, we set one body variable from each of $B_1, \ldots, B_\theta$ to false, the remaining variables in C to true and h to false. There are $|B_1| \times |B_2| \times \ldots \times |B_\theta| = O(n^\theta)$ such tuples. A3 now consists of all such tuples and the all-true tuple.

A3 acts like the search phase of the learning algorithm that looks for new universal Horn expressions(§3.2.1). A3 is a non-answer for q_i as at least one of the tuples has all variables in M set to true (because $M || B_i$ for $i = 1 \ldots \theta$) and h to false, thus violating $\forall M \rightarrow h$. □

LEMMA 4.7. *If h is a head variable in q_i but not in q_g then question A4 surfaces the difference.*

Proof: The all-true tuple satisfies all existential expressions in q_g. For each body variable x in q_g, A4 has a tuple where x is false and all other variables are true. If x is a head variable in q_i, then A4 should be a non-answer. □

This concludes the proof of Theorem. 4.2

5. RELATED WORK

Learning & Verifying Boolean Formula: Our work is influenced by the field of computational learning theory. Using membership questions to learn Boolean formulas was introduced in 1988 [3]. Angluin et al. demonstrated the polynomial learnability of conjunctions of (non-quantified) Horn clauses using membership questions and a more powerful class of questions known as equivalence questions [4]. The learning algorithm runs in time $O(k^2 n^2)$ where n is the number of variables and k is the number of clauses. Interestingly, Angluin proved that there is no PTIME algorithm for learning conjunctions of Horn clauses that only uses membership questions. Angluin et al.'s algorithm for learning conjunctions of Horn formula was extended to learn first-order Horn expressions [12, 10]. First-order Horn expressions contain quantifiers. We differ from this prior work in that in qhorn we quantify over tuples of an object's nested relation; we do not quantify over the values of variables. Our syntactic restrictions on qhorn have counterparts in Boolean formulas. Both qhorn-1 and *read-once* Boolean formulas [5] allow variables to occur at most once. Both role-preserving qhorn queries and *depth-1 acyclic Horn formulas* [9] do not allow variables to be both head and body variables.

Verification sets are analogous to the *teaching sequences* of Goldman and Kearns [8]. A teaching sequence is the smallest sequence of classified examples a teacher must reveal to a learner to help it uniquely identify a target concept from a concept class. Prior work provides algorithms to determine the teaching sequences for several classes of Boolean formula [6, 8, 14] but not for our class of qhorn queries.

Learning in the Database Domain: Two recent works on example-driven database query learning techniques — Query by Output (QBO) [17] and Synthesizing View Defi-

nitions (SVD) [7] — focus on the problem of learning a query Q from a given input database D, and an output view V. There are several key differences between this body of work and ours. First, QBO and SVD perform as decision trees; they infer a query's propositions so as to split D into tuples in V and tuples not in V. We assume that users can provide with us the propositions, so we focus on learning the structure of the query instead. Second, we work on a different subset of queries: QBO infers select-project-join queries and SVD infers unions of conjunctive queries. Learning unions of conjunctive queries is equivalent to learning k-term Disjunctive Normal Form (DNF) Boolean formulae [11]. We learn conjunctions of *quantified* Horn formulae. Since our target queries operate over objects with nested-sets of tuples instead of flat tuples, we learn queries in an exponentially larger query and data space. Finally, QBO and SVD work with a complete mapping from input tuples to output tuples. Our goal, however, is to learn queries from the smallest possible mapping of input to output objects, as it is generally impractical for users to label an entire database of objects as answers or non-answers. We point out that we synthesize our input when constructing membership questions, thus we can learn queries independent of the peculiarities of a particular input database D.

Using membership (and more powerful) questions to learn concepts within the database domain is not novel. For example, Cate, Dalmau and Kolaitis use membership and equivalence questions to learn schema mappings [16]. Staworko and Wieczorek use example XML documents given by the user to infer XML queries [15]. In both these works, the concept class learned is quite different from the qhorn query class.

6. CONCLUSION & FUTURE WORK

In this paper, we have studied the learnability of a special class of Boolean database queries — qhorn. We believe that other quantified-query classes (other than conjunctions of quantified Horn expressions) may exhibit different learnability properties. Mapping out the properties of different query classes will help us better understand the limits of example-driven querying. In our learning/verification model, we made the following assumptions: (i) the user's intended query is either in qhorn-1 or role-preserving qhorn, (ii) the data has at most one level nesting. We plan to design algorithms to verify that the user's query is indeed in qhorn-1 or role-preserving qhorn. We have yet to analyze the complexity of learning queries over data with multiple-levels of nesting. In such queries, a single expression can have several quantifiers.

We plan to investigate *Probably Approximately Correct* learning: we use randomly-generated membership questions to learn a query with a certain probability of error [18]. We note that membership questions provide only one bit of information — a response to membership question is either 'answer' (1) or 'non-answer' (0). We plan to examine the plausibility of constructing other types of questions that provide more information bits but still maintain interface usability. One possibility is to ask questions to directly determine how propositions interact[6] such as: "do you think p_1 and p_2 both have to be satisfied by at least one tuple?" or "when does p_1 have to be satisfied?"

[6]We thank our anonymous reviewer for this suggestion.

Finally, we see an opportunity to create efficient *query revision* algorithms. Given a query which is *close* to the user's intended query, our goal is to determine the intended query through few membership questions — polynomial in the distance between the given query and the intended query. Efficient revision algorithms exist for (non-quantified) role-preserving Horn formula [9]. The Boolean-lattice provides us with a natural way to measure how close two queries are: the distance between the distinguishing tuples of the given and intended queries.

7. ACKNOWLEDGMENTS

Partial funding provided by NSF Grants CCF-0963922, CCF-0916389, CC-0964033 and a Google University Research Award.

8. REFERENCES

[1] A. Abouzied et al. Learning and verifying quantified boolean queries by example. arXiv:1304.4303 [cs.DB].

[2] A. Abouzied, J. Hellerstein, and A. Silberschatz. Dataplay: interactive tweaking and example-driven correction of graphical database queries. In *UIST*, 2012.

[3] D. Angluin. Queries and concept learning. *Mach. Learn.*, 2(4):319–342, 1988.

[4] D. Angluin, M. Frazier, and L. Pitt. Learning conjunctions of horn clauses. In *COLT*, 1990.

[5] D. Angluin, L. Hellerstein, and M. Karpinski. Learning read-once formulas with queries. *J. ACM*, 40(1):185–210, 1993.

[6] M. Anthony et al. On exact specification by examples. In *COLT*, 1992.

[7] A. Das Sarma et al. Synthesizing view definitions from data. In *ICDT*, 2010.

[8] S. A. Goldman and M. J. Kearns. On the complexity of teaching. In *COLT*, 1991.

[9] J. Goldsmith and R. H. Sloan. New horn revision algorithms. *J. Mach. Learn. Res.*, 6:1919–1938, Dec. 2005.

[10] D. Haussler. Learning conjunctive concepts in structural domains. *Mach. Learn.*, 4(1):7–40, 1989.

[11] M. J. Kearns and U. V. Vazirani. *An introduction to computational learning theory.* MIT Press, Cambridge, MA, USA, 1994.

[12] R. Khardon. Learning first order universal horn expressions. In *COLT*, 1998.

[13] P. Reisner. Use of psychological experimentation as an aid to development of a query language. *IEEE Trans. on Soft. Eng.*, SE-3(3):218–229, 1977.

[14] A. Shinohara and S. Miyano. Teachability in computational learning. *New Gen. Comput.*, 8(4):337–347, 1991.

[15] S. Staworko and P. Wieczorek. Learning twig and path queries. In *ICDT*, 2012.

[16] B. ten Cate, V. Dalmau, and P. G. Kolaitis. Learning schema mappings. In *ICDT*, 2012.

[17] Q. T. Tran, C. Chan, and S. Parthasarathy. Query by output. In *SIGMOD*, 2009.

[18] L. G. Valiant. A theory of the learnable. *CACM*, 27(11):1134–1142, 1984.