This is the accepted manuscript of: ACM Transactions on Computation Theory 12 (1): 1-25 (2020)
https://doi.org/10.1145/3369930

New Steps on the Exact Learning of CNF

Montserrat Hermo[1] and Ana Ozaki[2]

[1]Languages and Information Systems, Univ. of the Basque Country, Spain
[2]Department of Computer Science, Univ. of Brasília, Brazil

January 30, 2024

Abstract

A major problem in computational learning theory is whether the class of formulas in conjunctive normal form (CNF) is efficiently learnable. Although it is known that this class cannot be polynomially learned using either membership or equivalence queries alone, it is open whether CNF can be polynomially learned using both types of queries. One of the most important results concerning a restriction of the class CNF is that propositional Horn formulas are polynomial time learnable in Angluin's exact learning model with membership and equivalence queries. In this work we push this boundary and show that the class of multivalued dependency formulas (MVDF) is polynomially learnable from interpretations. We then provide a notion of reduction between learning problems in Angluin's model, showing that a transformation of the algorithm suffices to efficiently learn multivalued database dependencies from data relations. We also show via reductions that our main result extends well known previous results and allows us to find alternative solutions for them.

1 Introduction

In the exact learning model, proposed by Angluin [2], a learner tries to identify an abstract target set by posing queries to an oracle. The most successful protocol uses membership and equivalence queries [20]. The exact learning model is distinguished by many other machine learning techniques for being a purely deductive reasoning approach. Since its proposal, a number of researchers have investigated which concept classes can be polynomially learned and it is known that algorithms in this model can be transformed into solutions for other well known settings such as the PAC [29, 3] and the online machine learning [24] models extended with membership queries.

Restrictions of CNF and DNF which have been proved to be polynomially learnable with membership and equivalence queries include: monotone DNF (DNF formulas with no negated variables) [3]; k-clause CNF (CNF formulas with at most k clauses) [1] and read-twice DNF (DNF where each variable occurs at most twice) [26]. The CDNF class (boolean functions whose CNF size is polynomial in its DNF size) [9] is also known to be learnable in polynomial time with both types of queries. Despite the intense effort to establish the complexity of learning the full classes of CNF and DNF, the complexity of these classes in the exact learning model with both queries remains open. It is known that these classes cannot be polynomially learned using either membership or equivalence queries alone [3, 4] and some advances in proving hardness of DNF with both queries appears in [15].

A classical result concerning a restriction of the class CNF appears in [5], where propositional Horn formulas are proved to be polynomially learnable with membership and equivalence queries. In fact, Horn is a special case of a class called k-quasi-Horn: clauses with at most k unnegated literals. However, it is pointed out by Angluin et. al [5] that, even for $k = 2$, learning the class of k-quasi-Horn formulas is as hard as learning CNF (Corollary 25 of [14]). Thus, if exact learning CNF is indeed intractable, the boundary of what can be learned in polynomial time with queries lies between 1-quasi-Horn (or simply Horn) and 2-quasi-Horn formulas. In this work we study the class of multivalued dependency formulas (MVDF) [28], which (as we explain in the Preliminaries) is a natural restriction of 2-quasi-Horn and a non-trivial generalization of Horn.

Another motivation to study the complexity of learning MVDF is that this class is the logical theory behind multivalued dependencies (MVD) [28, 8], in the sense that one can map a set of multivalued dependencies to a multivalued dependency formula preserving the logical consequence relation. A similar equivalence between functional dependencies and propositional Horn formulas is given by the authors of [10]. Although data dependencies are usually determined from the semantic attributes, they may not be known a priori by database designers. Discovering functional and multivalued dependencies from examples of data relations using inductive reasoning has been investigated by [18, 25, 17, 12]. Here we study this problem in Angluin's model. In this paper, we give a polynomial time algorithm that exactly learns multivalued dependencies formulas (MVDF) from interpretations. We then provide a formal notion of reduction for the exact learning model and use this notion to reduce the problem of learning MVD from data relations (and other problems below) to the problem of learning MVDF from interpretations.

Previous results. A large part of the related work was already mentioned. We now discuss some previous results which are extended by the present work. A polynomial time algorithm for exact learning (with membership and equivalence queries) propositional Horn from interpretations was first presented by Angluin et. al [5] (also, see [7]). One year later, Frazier and Pitt presented a polynomial time algorithm for exact learning propositional Horn from entailments [13]. More recently, Lavín proved polynomial time exact learnability of CRFMVF (resp., CRFMVD), which is a restriction of MVDF (resp., MVD) [23]. Then, a polynomial time algorithm for exact learning the full class MVDF from 2-quasi-Horn clauses was presented by the authors of [16].

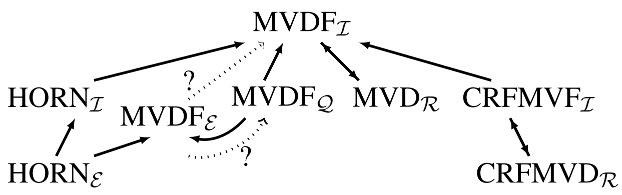

Figure 1: Reductions among learning problems

Figure 1 shows the relationship among learning problems via reductions, where $C_E \rightarrow C'_{E'}$ means that: the problem of exactly learning (with membership and equivalence queries) the class C from the examples E is reducible in polynomial time to the problem of exactly learning the class C' from E'. We use $\mathcal{I}$ for interpretations, $\mathcal{E}$ for entailments, $\mathcal{Q}$ for 2-quasi-Horn clauses and $\mathcal{R}$ for data relations. As shown in Figure 1, the problem MVDF$_{\mathcal{I}}$, solved in the present work, extends previous results on the efficient learnability of data dependencies and their corresponding propositional formulas. Our positive result for MVDF$_{\mathcal{I}}$ is a non-trivial extension of HORN$_{\mathcal{I}}$ (in [5]) and CRFMVF$_{\mathcal{I}}$ (in [23]) and allow us to prove for the first time the polynomial time learnability of the full class of multivalued dependencies from data relations (MVD$_{\mathcal{R}}$). As shown in Figure 1, one can reduce HORN$_{\mathcal{E}}$ to HORN$_{\mathcal{I}}$. However, we did not find a way of reducing MVDF$_{\mathcal{E}}$ to MVDF$_{\mathcal{I}}$ and we leave open the question of whether MVDF$_{\mathcal{E}}$ is polynomial time exactly learnable.

2 Preliminaries

Exact Learning Let E be a set of examples (also called *domain* or *instance space*). A *concept over* E is a subset of E and a *concept class* is a set C of concepts over E. Each concept c over E induces a dichotomy of *positive* and *negative* examples, meaning that $e \in c$ is a positive example and $e \in E \setminus c$ is a negative example. For computational purposes, concepts need to be specified by some representation. So we define a *learning framework* to be a triple $(E, \mathcal{L}, \mu)$, where E is a set of examples, $\mathcal{L}$ is a set of *concept representations* and μ is a surjective function from $\mathcal{L}$ to a concept class C of concepts over E.

Given a learning framework $\mathfrak{F} = (E, \mathcal{L}, \mu)$, for each $l \in \mathcal{L}$, denote by $\mathsf{MEM}_{l,\mathfrak{F}}$ the oracle that takes as input some $e \in E$ and returns 'yes' if $e \in \mu(l)$ and 'no' otherwise. A *membership query* is a call to an oracle $\mathsf{MEM}_{l,\mathfrak{F}}$ with some $e \in E$ as input, for $l \in \mathcal{L}$ and E. Similarly, for every $l \in \mathcal{L}$, we denote by $\mathsf{EQ}_{l,\mathfrak{F}}$ the oracle that takes as input a concept representation $h \in \mathcal{L}$ and returns 'yes', if $\mu(h) = \mu(l)$, or a *counterexample* $e \in \mu(h) \oplus \mu(l)$, otherwise. An *equivalence query* is a call to an oracle $\mathsf{EQ}_{l,\mathfrak{F}}$ with some $h \in \mathcal{L}$ as input, for $l \in \mathcal{L}$ and E. We say that a learning framework $(E, \mathcal{L}, \mu)$ is *exactly learnable* if there is an algorithm A such that for any target $l \in \mathcal{L}$ the algorithm A always halts and outputs $l' \in \mathcal{L}$ such that $\mu(l) = \mu(l')$ using membership and equivalence queries answered by the oracles $\mathsf{MEM}_{l,\mathfrak{F}}$ and

$\mathsf{EQ}_{l,\mathfrak{F}}$, respectively. A learning framework $(E, \mathcal{L}, \mu)$ is *polynomial time* exactly learnable if it is exactly learnable by a deterministic algorithm A such that at every step of computation the time used by A up to that step is bounded by a polynomial $p(|l|, |e|)$, where l is the target and $e \in E$ is the largest counterexample seen so far[1].

Multivalued Dependency Formulas Let V be a finite set of symbols, representing boolean variables. The logical constant *true* is represented by $\mathbf{T}$ and the logical constant *false* is represented by $\mathbf{F}$. A *multivalued* (for short *mvd*) *clause* is an implication $X \to Y \vee Z$, where X, Y and Z are pairwise disjoint conjunctions of variables from V and $X \cup Y \cup Z = V$. We note that some of X, Y, Z may be empty. An *mvd formula* is a conjunction of mvd clauses. A *k-quasi-Horn clause* is a propositional clause containing at most k unnegated literals. A *k-quasi-Horn formula* is a conjunction of k-quasi-Horn clauses. A *Horn clause* (resp., *Horn formula*) is a k-quasi-Horn clause (resp., *k-quasi-Horn formula*) with $k = 1$.
Remark: From the definition of an mvd clause and a k-quasi-Horn clause it is easy to see that:

1. any Horn clause is logically equivalent to a set of 2 mvd clauses. For instance, the Horn clause $135 \to 4$, is equivalent to: $\{12356 \to 4, 135 \to 4 \vee 26\}$;
2. any mvd clause is logically equivalent to a conjunction of 2-quasi-Horn clauses with size polynomial in the number of variables. For instance, the mvd clause $1 \to 23 \vee 456$, by distribution, is equivalent to: $\{1 \to 2 \vee 4, 1 \to 2 \vee 5, 1 \to 2 \vee 6, 1 \to 3 \vee 4, 1 \to 3 \vee 5, 1 \to 3 \vee 6\}$.

To simplify the notation, we treat sometimes conjunctions as sets and vice versa. Also, if for example $V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$ is a set of variables and $\varphi = (v_1 \to (v_2 \wedge v_3) \vee (v_4 \wedge v_5 \wedge v_6)) \wedge ((v_2 \wedge v_3) \to (v_1 \wedge v_5 \wedge v_6) \vee v_4)$ is a formula then we write φ in this shorter way: $\{1 \to 23 \vee 456, 23 \to 156 \vee 4\}$, where conjunctions between variables are omitted and each propositional variable $v_i \in V$ is mapped to $i \in \mathbb{N}$. For the purposes of this paper, we treat $X \to Y \vee Z$ and $X \to Z \vee Y$ as distinct mvd clauses, where Y and Z are non-empty. For example, $12 \to 34 \vee 56$ and $12 \to 56 \vee 34$ are counted as two distinct mvd clauses.

An interpretation $\mathcal{I}$ is a mapping from $V \cup \{\mathbf{T}, \mathbf{F}\}$ to $\{true, false\}$, where $\mathcal{I}(\mathbf{T}) = true$ and $\mathcal{I}(\mathbf{F}) = false$. We denote by $\mathsf{true}(\mathcal{I})$ the set of variables assigned to *true* in $\mathcal{I}$. In the same way, let $\mathsf{false}(\mathcal{I})$ be the set of variables assigned to *false* in $\mathcal{I}$. Observe that $\mathsf{false}(\mathcal{I}) = V \setminus \mathsf{true}(\mathcal{I})$. We follow the terminology provided by [5] and say that an interpretation $\mathcal{I}$ *covers* $X \to Y \vee Z$ if $X \subseteq \mathsf{true}(\mathcal{I})$. An interpretation $\mathcal{I}$ *violates* $X \to Y \vee Z$ if $\mathcal{I}$ *covers* $X \to Y \vee Z$ and: (a) Y and Z are non-empty and there are $v \in Y$ and $w \in Z$ such that $v, w \in \mathsf{false}(\mathcal{I})$; or (b) one of Y, Z is empty and there is $v \in Y \cup Z$ such that $\mathsf{false}(\mathcal{I}) = \{v\}$; or (c) $\mathsf{false}(\mathcal{I}) = \emptyset$ and $X \to Y \vee Z$ is the mvd clause $V \to \mathbf{F}$. If $\mathcal{I}$ does not violate $X \to Y \vee Z$ then we write $\mathcal{I} \models X \to Y \vee Z$.

[1] We count each call to an oracle as one step of computation. Also, we assume some natural notion of length for an example e and a concept representation l, denoted by $|e|$ and $|l|$, respectively.

Given two interpretations $\mathcal{I}$ and $\mathcal{I}'$, we define $\mathcal{I} \cap \mathcal{I}'$ to be the interpretation such that $\mathsf{true}(\mathcal{I} \cap \mathcal{I}') = \mathsf{true}(\mathcal{I}) \cap \mathsf{true}(\mathcal{I}')$. If $\mathcal{S}$ is a sequence of interpretations and $\mathcal{I}$ is an interpretation occurring at position i then we write $\mathcal{I}_i \in \mathcal{S}$. Also, we denote by $\mathcal{S} \cdot \mathcal{I}$ the result of appending $\mathcal{I}$ to $\mathcal{S}$. The learning MVDF from interpretations framework is defined as $\mathfrak{F}(\text{MVDF}_{\mathcal{I}}) = (E_{\mathcal{I}}, \mathcal{L}_{\mathsf{M}}, \mu_{\mathcal{I}})$, where $E_{\mathcal{I}}$ is the set of all interpretations in the propositional variables V under consideration, $\mathcal{L}_{\mathsf{M}}$ is the set of all sets of mvd clauses that can be expressed in V and, for every $\mathcal{T} \in \mathcal{L}_{\mathsf{M}}$, $\mu_{\mathcal{I}}(\mathcal{T}) = \{\mathcal{I} \in E_{\mathcal{I}} \mid \mathcal{I} \models \mathcal{T}\}$.

3 Learning MVDF from Interpretations

In this section we present an algorithm that learns the class MVDF from interpretations. More precisely, we show that the learning framework $\mathfrak{F}(\text{MVDF}_{\mathcal{I}})$ is polynomial time exactly learnable.

The learning algorithm for $\mathfrak{F}(\text{MVDF}_{\mathcal{I}})$ is given by Algorithm 1. Algorithm 1 maintains a sequence $\mathfrak{P}$ of interpretations which are positive examples for the target $\mathcal{T}$ and a sequence $\mathfrak{L}$ of interpretations which are negative examples (for the target $\mathcal{T}$). The learner's hypothesis $\mathcal{H}$ is constructed using both $\mathfrak{P}$ and $\mathfrak{L}$. In order to learn all of the mvd clauses in $\mathcal{T}$, we would like that mvd clauses induced by the elements of $\mathfrak{P}$ and $\mathfrak{L}$ approximate distinct mvd clauses in $\mathcal{T}$. This will happen if at most polynomially many elements in $\mathfrak{L}$ violate the same mvd clause in $\mathcal{T}$. Overzealous refinement of a sequence of interpretations is a situation described by [5]. It may result in a loop where we have several elements of the sequence violating the same clause in the target. We avoid this in Algorithm 1 by (1) refining negative counterexamples with elements of $\mathfrak{L}$ (Line 9) and (2) refining at most one (the first) element of $\mathfrak{L}$ per iteration (Line 13). We use the following notion, provided by [16], to describe under which conditions the learner should refine either a negative counterexample or an element of $\mathfrak{L}$.

Definition 1 *A pair* $(\mathcal{I}, \mathcal{I}')$ *of interpretations is a* $\mathsf{goodCandidate}$ *if: (i)* $\mathsf{true}(\mathcal{I} \cap \mathcal{I}') \subset \mathsf{true}(\mathcal{I})$*; (ii)* $\mathcal{I} \cap \mathcal{I}' \models \mathcal{H}$*; and (iii)* $\mathcal{I} \cap \mathcal{I}' \not\models \mathcal{T}$.

In the following we provide the main ideas of our proof (omitted proofs are given in full detail in the appendix). If Algorithm 1 terminates, then it obviously has found a hypothesis $\mathcal{H}$ that is logically equivalent to $\mathcal{T}$, formulated with variables in V. It thus remains to show that Algorithm 1 terminates in polynomial time. In each iteration, one of the following three cases happens:

1. a positive counterexample is added to the sequence $\mathfrak{P}$ (Line 7); or

2. a negative example in $\mathfrak{L}$ is replaced (Line 13); or

3. a negative counterexample is appended to the sequence $\mathfrak{L}$ (Line 16).

To prove polynomial time learnability, we need to ensure that each iteration is done in polynomial time in the size of $\mathcal{T}$ and that the total number of iterations is

Algorithm 1 Learning algorithm for MVDF from Interpretations

1: Let $\mathfrak{L}$ be a sequence of negative examples and $\mathfrak{P}$ a sequence of positive examples
2: Set $\mathcal{H}_0 := \{V \to \mathbf{F} \mid \mathcal{T} \models V \to \mathbf{F}\} \cup \{V \setminus \{v\} \to v \mid v \in V$ and $\mathcal{T} \models V \setminus \{v\} \to v\}$
3: Set $\mathfrak{L} := \emptyset$, $\mathfrak{P} := \emptyset$ and $\mathcal{H} := \mathcal{H}_0$
4: **while** $\mathcal{H} \not\equiv \mathcal{T}$ **do**
5:     Let $\mathcal{I}$ be a counterexample
6:     **if** $\mathcal{I} \not\models \mathcal{H}$ **then**
7:         Append $\mathcal{I}$ to $\mathfrak{P}$
8:     **else**
9:         Set $\mathcal{J} :=$ REFINECOUNTEREXAMPLE($\mathcal{I}$,$\mathfrak{L}$)
10:         **if** there is $\mathcal{I}_k \in \mathfrak{L}$ such that $\mathsf{goodCandidate}(\mathcal{I}_k, \mathcal{J})$ **then**
11:             Let $\mathcal{I}_i$ be the first in $\mathfrak{L}$ such that $\mathsf{goodCandidate}(\mathcal{I}_i, \mathcal{J})$
12:             Set $\mathfrak{P}' :=$ UPDATEPOSITIVEEXAMPLES($\mathcal{J}, \mathfrak{P}, \mathfrak{L}$) and $\mathfrak{P} := \mathfrak{P}'$
13:             Replace $\mathcal{I}_i \in \mathfrak{L}$ by $\mathcal{J}$
14:             Remove all $\mathcal{I}_j \in \mathfrak{L} \setminus \{\mathcal{J}\}$ such that $\mathcal{I}_j \not\models$BUILDCLAUSES($\mathcal{J}, \mathfrak{P}$)
15:         **else**
16:             Append $\mathcal{J}$ to $\mathfrak{L}$
17:         **end if**
18:     **end if**
19:     Construct $\mathcal{H} := \mathcal{H}_0 \cup \bigcup_{\mathcal{I} \in \mathfrak{L}}$BUILDCLAUSES($\mathcal{I}, \mathfrak{P}$)
20: **end while**

also bounded. That is, the number of times Cases 1, 2 and 3 happen is polynomial in the size of $\mathcal{T}$. For Case 2 we note that each time a negative example is replaced, the number of variables assigned to true strictly decreases (Point (i) of Definition 1). Then, Algorithm 1 replaces each element of $\mathfrak{L}$ at most $|V|$ times.

Before we give a bound for Cases 1 and 3, we explain the bound on the number of recursive calls. We first note that in each recursive call of Function 'RefineCounterexample' (Algorithm 2) the number of variables assigned to true in a negative counterexample strictly decreases (Point (i) of Definition 1). This means that in each iteration of Algorithm 1 the number of recursive calls of Function 'RefineCounterexample' is at most $|V|$. To see the bound on the number of recursive calls of Function 'UpdatePositiveExamples' (Algorithm 4) we use Lemma 2. By construction of $\mathcal{H}_0$ (Line 2 of Algorithm 1) we can assume that all negative examples we deal with violate $X \to Y \vee Z \in \mathcal{T}$ with Y, Z non-empty[2]. We write BUILDCLAUSES($\mathcal{I}, \mathfrak{P}$) for the set of mvd clauses returned as output of Function 'BuildClauses' (Algorithm 3) with $\mathcal{I}$ and $\mathfrak{P}$ as input.

Lemma 2 *Let $\mathcal{I}$ be a negative example. Let* BUILDCLAUSES*($\mathcal{I}, \mathfrak{P}$)* $= \{\mathsf{true}(\mathcal{I}) \to$

[2]We note that one can easily check whether '$\mathcal{T} \models V \to \mathbf{F}$' and '$\mathcal{T} \models V \setminus \{v\} \to v$' with membership queries that receive interpretations as input.

Algorithm 2 Function RefineCounterexample $(\mathcal{I}, \mathfrak{L})$

Set $\mathcal{J} := \mathcal{I}$
if there is $\mathcal{I}_k \in \mathfrak{L}$ such that $\mathsf{goodCandidate}(\mathcal{I}, \mathcal{I}_k)$ **then**
  Let $\mathcal{I}_i$ be the first in $\mathfrak{L}$ such that $\mathsf{goodCandidate}(\mathcal{I}, \mathcal{I}_i)$
  $\mathcal{J}$:=REFINECOUNTEREXAMPLE($\mathcal{I} \cap \mathcal{I}_i$,$\mathfrak{L}$)
end if
return$(\mathcal{J})$

$Y_1 \vee Z_1, \ldots, \mathsf{true}(\mathcal{I}) \rightarrow Y_k \vee Z_k\}$. *Then, for all* i, j, *such that* $1 \leq i < j \leq k$, *we have that:* $Y_i \cap Y_j = \emptyset$ *and* $\bigcup_{j=1}^{k} Y_j = \mathsf{false}(\mathcal{I})$. *Moreover, for any* $\mathsf{true}(\mathcal{I}) \rightarrow Y_i \vee Z_i$, $1 \leq i \leq k$, *we have that* Y_i, Z_i *are non-empty.*

Algorithm 3 Function BuildClauses $(\mathcal{I}, \mathfrak{P})$

Set $X := \mathsf{true}(\mathcal{I})$ and $C := \{X \rightarrow v \vee V \setminus (X \cup \{v\}) \mid v \in V \setminus X\}$
for each $\mathcal{I}_l \in \mathfrak{P}$ **do**
  Let $X \rightarrow Y_1 \vee Z_1, \ldots, X \rightarrow Y_k \vee Z_k$ be the mvd clauses in C violated by $\mathcal{I}_l$
  Replace in C all these mvd clauses by $X \rightarrow \bigcup_{j=1}^{k} Y_j \vee (\bigcup_{j=1}^{k} Z_j \setminus \bigcup_{j=1}^{k} Y_j)$
end for
return (C)

Algorithm 4 Function UpdatePositiveExamples $(\mathcal{K}, \mathfrak{P}, \mathfrak{L})$

Set $\mathfrak{P}' := \mathfrak{P}$
if there are distinct $\mathcal{I}_k, \mathcal{I}_l \in \mathfrak{L}$ such that $\mathcal{I}_k \cap \mathcal{I}_l \not\models$BUILDCLAUSES($\mathcal{K}, \mathfrak{P}$) and $\mathcal{I}_k \cap \mathcal{I}_l \models \mathcal{T}$ **then**
  Append $\mathcal{I}_k \cap \mathcal{I}_l$ to $\mathfrak{P}$
  $\mathfrak{P}'$:=UPDATEPOSITIVEEXAMPLES($\mathcal{K}, \mathfrak{P}, \mathfrak{L}$)
end if
return $(\mathfrak{P}')$

By Lemma 2 above we have that the 'Y' consequents of mvd clauses returned by Function 'BuildClauses' (Algorithm 3) are non-empty and mutually disjoint. So the number of mvd clauses returned by this function is bounded by $|V|$. Regarding Function 'UpdatePositiveExamples' (Algorithm 4) called in Line 12, we note that $\mathcal{K} = \mathcal{J}$ is a negative example and that in Line 3, we have that $\mathcal{I}_k \cap \mathcal{I}_l \not\models$BUILDCLAUSES($\mathcal{K}, \mathfrak{P}$). Then, the next lemma ensures that in each recursive call of Function 'UpdatePositiveExamples' (Algorithm 4) the number of mvd clauses returned by Function 'BuildClauses' (Algorithm 3) with $\mathcal{K}$ and $\mathfrak{P}$ as input, strictly decreases. Since (by Lemma 2 above) the number of mvd clauses returned by Function 'BuildClauses' (Algorithm 3) is at most $|V|$, the next lemma bounds the number of recursive calls of Function 'UpdatePositiveExamples' (Algorithm 4) to $|V|$.

Lemma 3 *Let $\mathcal{I}$ be a negative example. If $\mathcal{P} \not\models$ BUILDCLAUSES$(\mathcal{I}, \mathfrak{P})$ then the number of mvd clauses returned by* BUILDCLAUSES$(\mathcal{I}, \mathfrak{P} \cdot \mathcal{P})$ *is strictly smaller than the number of mvd clauses returned by* BUILDCLAUSES$(\mathcal{I}, \mathfrak{P})$.

Proof. Suppose that $\mathsf{true}(\mathcal{I}) \rightarrow Y_i \vee Z_i \in$ BUILDCLAUSES$(\mathcal{I}, \mathfrak{P})$ is violated by $\mathcal{P}$. Then, there is $v \in Y_i$ and $w \in Z_i$ such that $v, w \in \mathsf{false}(\mathcal{P})$. By Lemma 2 there is $\mathsf{true}(\mathcal{I}) \rightarrow Y_j \vee Z_j \in$ BUILDCLAUSES$(\mathcal{I}, \mathfrak{P})$ such that $w \in Y_j$ and $v \in Z_j$. In Line 4, Algorithm 3 replaces (at least) these two mvd clauses by a single mvd clause. So the number of mvd clauses strictly decreases, as required. ❑

By Lemma 4 below if any two interpretations $\mathcal{I}_i, \mathcal{I}_j \in \mathfrak{L}$ violate the same mvd clause in $\mathcal{T}$ then their sets of false variables are mutually disjoint. By construction of $\mathcal{H}_0$ we can assume that their sets of false variables are non-empty. Then, the number of interpretations violating any mvd clause in $\mathcal{T}$ is bounded by $|V|$.

Lemma 4 *Let $\mathcal{I}_i, \mathcal{I}_j \in \mathfrak{L}$ and assume $i < j$. At the end of each iteration, if $\mathcal{I}_i, \mathcal{I}_j \in \mathfrak{L}$ violate $c \in \mathcal{T}$ then $\mathsf{false}(\mathcal{I}_i) \cap \mathsf{false}(\mathcal{I}_j) = \emptyset$.*

Corollary 5 *At the end of each iteration every $c \in \mathcal{T}$ is violated by at most $|V|$ interpretations in $\mathfrak{L}$.*

So, at all times the number of elements in $\mathfrak{L}$ is bounded by $|\mathcal{T}| \cdot |V|$. We now show that the number of iterations of Algorithm 1 is polynomial in the size of $\mathcal{T}$. We first present in Lemma 7 a polynomial upper bound on the number of iterations where Algorithm 1 receives a negative counterexample. Note that we obtain this upper bound even though the learner does not know the size $|\mathcal{T}|$ of the target. Lemma 7 requires the following technical lemma.

Lemma 6 *In Line 14 of Algorithm 1, the following holds:*

1. *if $\mathcal{I}_j$ is removed after the replacement of some $\mathcal{I}_i \in \mathfrak{L}$ by $\mathcal{J}$ (Line 13) then $\mathsf{false}(\mathcal{I}_i) \cap \mathsf{false}(\mathcal{I}_j) = \emptyset$ ($\mathcal{I}_i$ before the replacement);*
2. *if $\mathcal{I}_j, \mathcal{I}_k$ with $j < k$ are removed after the replacement of some $\mathcal{I}_i \in \mathfrak{L}$ by $\mathcal{J}$ (Line 13) then $\mathsf{false}(\mathcal{I}_j) \cap \mathsf{false}(\mathcal{I}_k) = \emptyset$.*

Lemma 7 *Let N be $|V|^2 \cdot |\mathcal{T}|$. The expression $E = |\mathfrak{L}| + (N - \sum_{\mathcal{I} \in \mathfrak{L}} |\mathsf{false}(\mathcal{I})|)$ always evaluates to a natural number inside the loop body and decreases on every iteration where Algorithm 1 receives a negative counterexample.*

Proof. By Corollary 5, the size of $\mathfrak{L}$ is bounded at all times by $|V| \cdot |\mathcal{T}|$. Thus, by Corollary 5, N is an upper bound for $\sum_{\mathcal{I} \in \mathfrak{L}} |\mathsf{false}(\mathcal{I})|$. If a negative counterexample is received then there are three possibilities: (1) an element $\mathcal{I}$ is appended to $\mathfrak{L}$. Then, $|\mathfrak{L}|$ increases by one but $|\mathsf{false}(\mathcal{I})| \geq 2$ and, therefore, E decreases; (2) an element is replaced and no element is removed. Then, E trivially decreases. Otherwise, (3) we have that an element $\mathcal{I}_i$ is replaced and p interpretations are removed from $\mathfrak{L}$ in Line 14 of Algorithm 1. By Point 2 of Lemma 6, if $\mathcal{I}_i$ is replaced by $\mathcal{J}$ and $\mathcal{I}_j, \mathcal{I}_k$ are removed then $\mathsf{false}(\mathcal{I}_j) \cap \mathsf{false}(\mathcal{I}_k) = \emptyset$. This means that if p interpretations are removed then their sets of false variables are all mutually disjoint. By Point 1 of Lemma 6, if $\mathcal{I}_i$ is replaced by $\mathcal{J}$ and some $\mathcal{I}_j$ is removed then $\mathsf{false}(\mathcal{I}_j) \cap \mathsf{false}(\mathcal{I}_i) = \emptyset$. Then, the p interpretations also have sets of false variables disjoint from $\mathsf{false}(\mathcal{I}_i)$. For each interpretation $\mathcal{I}_j$ removed we have $\mathsf{false}(\mathcal{I}_j) \subseteq \mathsf{false}(\mathcal{J})$ (because $\mathcal{I}_j \not\models$BUILDCLAUSES($\mathcal{J}$,$\mathfrak{P}$). Then, the number of 'falses' is at least as large as before. However $|\mathfrak{L}|$ decreases and, thus, we can ensure that E decreases. ❑

By Lemma 7 the total number of iterations where Algorithm 1 receives a negative counterexample is bounded by $N = |V|^2 \cdot |\mathcal{T}|$. It remains to show a polynomial bound on the total number of iterations where Algorithm 1 receives a positive counterexample. By Corollary 5, the size of $\mathfrak{L}$ is bounded at all times by $|V| \cdot |\mathcal{T}|$. By Lemma 2, the number of clauses induced by each $\mathcal{I}_i \in \mathfrak{L}$ is bounded by $|V|$. This means that the size of $\mathcal{H}$ is bounded at all times by N. If a positive counterexample is received then, by Lemma 3, the size of $\mathcal{H}$ strictly decreases. So after giving at most $|\mathcal{H}| \leq N$ positive examples the oracle is forced to give a negative example. Since the number of negative counterexamples received is also bounded by N, the total number of iterations where Algorithm 1 receives a positive counterexample is bounded by N^2.

Theorem 8 *The problem of learning MVDF from interpretations, more precisely, the learning framework $\mathfrak{F}$(MVDF$_{\mathcal{I}}$), is polynomial time exactly learnable.*

3.1 An Example Run

We describe an example run of Algorithm 1. In this example, if Function 'BuildClauses' (Algorithm 3) returns as output mvd clauses of the form $X \to Y \vee Z$ and $X \to Z \vee Y$ then we write only one of them. Suppose that our target MVDF is:

$$\mathcal{T} = \{2345 \to 1, \quad 123 \to 4 \vee 5, \quad 235 \to 1 \vee 4, \quad 2 \to 3 \vee 145\}.$$

Initially, the sequence $\mathfrak{P}$ of positive examples and the sequence $\mathfrak{L}$ of negative examples are both empty. In Line 2 of Algorithm 1, we construct $\mathcal{H}_0 = \{2345 \to 1\}$. Suppose that the counterexample to our first equivalence query with $\mathcal{H} = \mathcal{H}_0$ as input is the negative example $\mathcal{I}_1$, with $\mathsf{true}(\mathcal{I}_1) = \{1, 2, 3\}$ (note that $\mathcal{I}_1$ violates the second mvd clause in $\mathcal{T}$). Since $\mathfrak{L}$ is empty, Algorithm 1 simply appends $\mathcal{I}_1$ to $\mathfrak{L}$. In Line 19, Algorithm 1 calls Function 'BuildClauses' (Algorithm 3) with $\mathcal{I}_1, \mathfrak{P}$

as input and receive $\{123 \to 4 \vee 5\}$ as output. At this moment, $\mathfrak{P}$, $\mathfrak{L}$ and $\mathcal{H}$ are as follows.

$$\mathfrak{P} = \emptyset \quad \mathfrak{L} = \{\mathcal{I}_1\} \quad \mathcal{H} = \{2345 \to 1, 123 \to 4 \vee 5\}$$

Suppose that the counterexample to our second equivalence query with $\mathcal{H}$ as input is $\mathcal{I}_2$, with $\mathsf{true}(\mathcal{I}_2) = \{2, 3, 5\}$. Since $\mathcal{I}_2 \cap \mathcal{I}_1$ satisfies $\mathcal{T}$, the pair $(\mathcal{I}_2, \mathcal{I}_1)$ is not a $\mathsf{goodCandidate}$. So Algorithm 1 appends $\mathcal{I}_2$ to $\mathfrak{L}$. In Line 19, Algorithm 1 calls Function 'BuildClauses' (Algorithm 3) with $\mathcal{I}_1, \mathfrak{P}$ and $\mathcal{I}_2, \mathfrak{P}$ as inputs. We have that $\mathfrak{P}$, $\mathfrak{L}$ and $\mathcal{H}$ are as follows.

$$\mathfrak{P} = \emptyset \quad \mathfrak{L} = \{\mathcal{I}_1, \mathcal{I}_2\} \quad \mathcal{H} = \{2345 \to 1, 123 \to 4 \vee 5, 235 \to 1 \vee 4\}$$

Now assume that the next counterexample is $\mathcal{I}_3$, with $\mathsf{true}(\mathcal{I}_3) = \{2, 4\}$. In Line 9, Algorithm 1 calls Function 'RefineCounterexample' (Algorithm 2) with $\mathcal{I}_3$ and $\mathfrak{L}$ as input and verifies that the pair $(\mathcal{I}_3, \mathcal{I}_1)$ is a $\mathsf{goodCandidate}$. The return of Function 'RefineCounterexample' (Algorithm 2) is $\mathcal{J} = (\mathcal{I}_3 \cap \mathcal{I}_1)$. In Line 10, Algorithm 1 verifies that $\mathcal{I}_1$ is the first element in $\mathfrak{L}$ such that $(\mathcal{I}_1, \mathcal{J})$ is a $\mathsf{goodCandidate}$. Then, Algorithm 1 calls Function 'UpdatePositiveExamples' (Algorithm 4) with $\mathcal{K} = \mathcal{J}$ (note that $\mathsf{true}(\mathcal{K}) = \{2\}$), $\mathfrak{P}$ and $\mathfrak{L}$ as input. We have that

$$\text{BUILDCLAUSES}(\mathcal{K}, \emptyset) = \{2 \to 1 \vee 345, 2 \to 3 \vee 145, 2 \to 4 \vee 135, 2 \to 5 \vee 134\}.$$

As $(\mathcal{I}_1 \cap \mathcal{I}_2) \not\models \text{BUILDCLAUSES}(\mathcal{K}, \emptyset)$ and $(\mathcal{I}_1 \cap \mathcal{I}_2) \models \mathcal{T}$, the condition in Line 2 of Function 'UpdatePositiveExamples' (Algorithm 4) is satisfied. Then, Function 'UpdatePositiveExamples' appends $\mathcal{I}_1 \cap \mathcal{I}_2$ to $\mathfrak{P}$ and makes a recursive call with $\mathcal{K}$, $\mathfrak{P}$ and $\mathfrak{L}$ as input. Now,

$$\text{BUILDCLAUSES}(\mathcal{K}, \{\mathcal{I}_1 \cap \mathcal{I}_2\}) = \{2 \to 145 \vee 3\},$$

and, so, $(\mathcal{I}_1 \cap \mathcal{I}_2) \models \text{BUILDCLAUSES}(\mathcal{K}, \{\mathcal{I}_1 \cap \mathcal{I}_2\})$. The output of Function 'UpdatePositiveExamples' (Algorithm 4) is $\{\mathcal{I}_1 \cap \mathcal{I}_2\}$. In Line 13, Algorithm 1 replaces $\mathcal{I}_1 \in \mathfrak{L}$ by $\mathcal{J}$. In Line 19, Algorithm 1 calls Function 'BuildClauses' (Algorithm 3) with $\mathcal{J}, \mathfrak{P}$ and $\mathcal{I}_2, \mathfrak{P}$ as inputs. Now, $\mathfrak{P}$, $\mathfrak{L}$ and $\mathcal{H}$ are as follows.

$$\mathfrak{P} = \{\mathcal{I}_1 \cap \mathcal{I}_2\} \quad \mathfrak{L} = \{\mathcal{J}, \mathcal{I}_2\} \quad \mathcal{H} = \{2345 \to 1, 2 \to 145 \vee 3, 235 \to 1 \vee 4\}$$

Now assume that the counterexample to our fourth equivalence query with $\mathcal{H}$ as input is the negative example $\mathcal{I}_4$, with $\mathsf{true}(\mathcal{I}_4) = \{1, 2, 3\}$. Function 'RefineCounterexample' (Algorithm 2) returns $\mathcal{I}_4$. Since there is no $\mathcal{I} \in \mathfrak{L}$ such that $(\mathcal{I}, \mathcal{I}_4)$ is a $\mathsf{goodCandidate}$, Algorithm 1 appends $\mathcal{I}_4$ to $\mathfrak{L}$. In Line 19 of Algorithm 1 $\mathfrak{P}$, $\mathfrak{L}$ and $\mathcal{H}$ are as follows.

$$\mathfrak{P} = \{\mathcal{I}_1 \cap \mathcal{I}_2\} \quad \mathfrak{L} = \{\mathcal{J}, \mathcal{I}_2, \mathcal{I}_4\}$$

$$\mathcal{H} = \{2345 \to 1, 2 \to 145 \vee 3, 235 \to 1 \vee 4, 123 \to 4 \vee 5\}$$

We now have that $\mathcal{H} \equiv \mathcal{T}$ and the learner succeeded.

4 Reductions among Learning Problems

A substitution-based technique for problem reductions among boolean formulas was presented by [19]. [27] define a general type of problem reduction that preserves polynomial time prediction. This notion was extended by [6] to allow membership queries. In this section, we present a notion of reduction that is suitable for the exact learning model with membership and equivalence queries. It extends a notion of reduction given by [21]. We then use this notion to show the reductions in Figure 1.

Suppose that P is the problem of exactly learning the framework $\mathfrak{F} = (X, \mathcal{L}, \mu)$ and P' is the problem of exactly learning the framework $\mathfrak{F}' = (X', \mathcal{L}, \mu')$. Since $\mathcal{L}$ is the same for $\mathfrak{F}$ and $\mathfrak{F}'$, every correct conjecture used to solve P' is also an answer for P and vice-versa. One can then reduce P to P' by: (a) transforming queries posed to oracles $\mathsf{MEM}_{l,\mathfrak{F}'}$ and $\mathsf{EQ}_{l,\mathfrak{F}'}$ into queries for the oracles $\mathsf{MEM}_{l,\mathfrak{F}}$ and $\mathsf{EQ}_{l,\mathfrak{F}}$; and (b) transforming answers given by the oracles $\mathsf{MEM}_{l,\mathfrak{F}}$ and $\mathsf{EQ}_{l,\mathfrak{F}}$ into answers that the oracles $\mathsf{MEM}_{l,\mathfrak{F}'}$ and $\mathsf{EQ}_{l,\mathfrak{F}'}$ would provide, where $l \in \mathcal{L}$ is the learning target. For our purposes, we want reductions where (i) the frameworks use the same target concept representation (as described above) and (ii) preserve polynomial time exact learnability. We say that a learning framework $\mathfrak{F} = (E, \mathcal{L}, \mu)$ *polynomial time reduces* to $\mathfrak{F}' = (E', \mathcal{L}, \mu')$ if, for some polynomials $p_1(\cdot)$, $p_2(\cdot, \cdot)$ and $p_3(\cdot, \cdot)$ there exist a function $f_{\mathsf{MEM}} : \mathcal{L} \times E' \to \{$ 'yes', 'no'$\}$, translating a $\mathfrak{F}'$ membership query to $\mathfrak{F}$, and a partial function $f_{\mathsf{EQ}} : \mathcal{L} \times \mathcal{L} \times E \to E'$, defined for every (l, h, e) such that $|h| \leq p_1(|l|)$, translating an answer to an $\mathfrak{F}$ equivalence query to $\mathfrak{F}'$, for which the following conditions hold:

- for all $e' \in E'$ we have $e' \in \mu'(l)$ iff $f_{\mathsf{MEM}}(l, e') =$ 'yes';
- for all $e \in E$ we have $e \in \mu(l) \oplus \mu(h)$ iff $f_{\mathsf{EQ}}(l, h, e) \in \mu'(l) \oplus \mu'(h)$;
- $f_{\mathsf{MEM}}(l, e')$ and $f_{\mathsf{EQ}}(l, h, e)$ are computable in time $p_2(|l|, |e'|)$ and $p_3(|l|, |e|)$, respectively, and l can only be accessed by calls to the membership oracle $\mathsf{MEM}_{l,\mathfrak{F}}$.

Note that even though f_{EQ} takes h as input, the polynomial time bound on computing $f_{\mathsf{EQ}}(l, h, e)$ does not depend on the size of h as f_{EQ} is only defined for h polynomial in the size of l.

Theorem 9 *Let $\mathfrak{F} = (E, \mathcal{L}, \mu)$ and $\mathfrak{F}' = (E', \mathcal{L}, \mu')$ be learning frameworks. If there exists a polynomial time reduction from $\mathfrak{F}$ to $\mathfrak{F}'$ and $\mathfrak{F}'$ is polynomial time exactly learnable then $\mathfrak{F}$ is polynomial time exactly learnable.*

In the following we use Theorem 9 to prove that MVD can be learned in polynomial time from data relations. The remaining reductions presented in Figure 1 are given in the appendix.

4.1 Learning MVD from Data Relations

Notation A *relation scheme* $V = \{A_1, \ldots, A_n\}$ is a finite set of symbols, called attributes, where each attribute $A_i \in V$ is associated with a domain $\mathsf{dom}(A_i)$ of values. A *tuple* t over V is an element of $\mathsf{dom}(A_1) \times \ldots \times \mathsf{dom}(A_n)$. A *relation* r (over V) is a set of tuples over V. Given $S \subseteq V$, let $t[S]$ denote the restriction of a tuple t over V on S. For example, if the relation scheme is $\mathtt{PERSON} = \{\mathtt{NAME}, \mathtt{BOOK}, \mathtt{PET}\}$ and $t = (\mathtt{Alice}, \mathtt{Hamlet}, \mathtt{Dog})$ is a tuple over $\mathtt{PERSON}$ then $t[\{\mathtt{NAME}, \mathtt{PET}\}] = (\mathtt{Alice}, \mathtt{Dog})$. Let X, Y and Z be pairwise disjoint subsets of V with $X \cup Y \cup Z = V$. We write xyz for a tuple t over V with $t[X] = x$, $t[Y] = y$ and $t[Z] = z$. A *multivalued dependency* (for short *mvd*) $X \to Y \vee Z$ holds in r if, and only if, for each two tuples $xyz, xy'z' \in r$ we have that $xy'z \in r$ (and, by symmetry, $xyz' \in r$) [3]. That is, if t, t' are distinct tuples in r with $t[X] = t'[X]$ then we can exchange the Y values of t, t' to obtain two tuples that must also be in r. If $\mathcal{T}$ is a set of mvds over V and, for all $m \in \mathcal{T}$, m holds in r (over V) then we say that $\mathcal{T}$ holds in r. We formally define the learning framework $\mathfrak{F}$(MVD$_\mathcal{R}$) as $(E_\mathcal{R}, \mathcal{L}_\mathsf{M}, \mu_\mathcal{R})$, where $E_\mathcal{R}$ is the set of all relations r over a relation scheme V, $\mathcal{L}_\mathsf{M}$ is the set of all sets of mvds that can be expressed with symbols in V and, for every $\mathcal{T} \in \mathcal{L}_\mathsf{M}$, $\mu_\mathcal{R}(\mathcal{T}) = \{r \in E_\mathcal{R} \mid \mathcal{T} \text{ holds in } r\}$.

We now show that $\mathfrak{F}$(MVD$_\mathcal{R}$) polynomial time reduces to $\mathfrak{F}$(MVDF$_\mathcal{I}$). To reduce the problem, we use the learning algorithm for $\mathfrak{F}$(MVDF$_\mathcal{I}$) as a 'black box' and: (1) transform the membership queries, which come as interpretations into relations; and (2) transform counterexamples given by equivalence queries, which come as relations into interpretations.

Lemma 10 *Let $\mathfrak{F}$(MVD$_\mathcal{R}$) $= (E_\mathcal{R}, \mathcal{L}_\mathsf{M}, \mu_\mathcal{R})$ and $\mathfrak{F}$(MVDF$_\mathcal{I}$) $= (E_\mathcal{I}, \mathcal{L}_\mathsf{M}, \mu_\mathcal{I})$ be, respectively, the frameworks for learning MVD from relations and learning MVDF from interpretations. Let $\mathcal{T} \in \mathcal{L}_\mathsf{M}$ be the target. For any interpretation $\mathcal{I} \in \mu_\mathcal{I}(\mathcal{T})$, one can construct in polynomial time in $|\mathcal{T}|$ a relation r such that $r \in \mu_\mathcal{R}(\mathcal{T})$ if, and only if, $\mathcal{I} \in \mu_\mathcal{I}(\mathcal{T})$.*

Proof. Given an interpretation $\mathcal{I}$ in V, we define a pair p of tuples $\{t, t'\}$ over V such that, for each $\gamma \in V$, $t[\gamma] = t'[\gamma]$ if, and only if, $\gamma \in \mathsf{true}(\mathcal{I})$. By definition of p, we have that, for any $m \in \mathcal{T}$, m does not hold in p if, and only if, $\mathcal{I}$ violates m. Then, $p \in \mu_\mathcal{R}(\mathcal{T})$ if, and only if, $\mathcal{I} \in \mu_\mathcal{I}(\mathcal{T})$. ❏

The close connection between database relations and propositional logic interpretations was first pointed out by [10] and its use in a learning theory context appears in [22]. To show Lemma 12 we use the following technical lemma, given by [28].

[3] The standard notation used for mvds is $X \twoheadrightarrow Y \mid Z$ (or $X \twoheadrightarrow Y$) [11]. However, for the purpose of showing a reduction from MVD$_\mathcal{R}$ to MVDF$_\mathcal{I}$, it is useful to adopt a uniform representation between the two classes.

Lemma 11 ([28]) *Assume that r is a relation over V, $\mathcal{T}$ is a set of mvds and m is an mvd (both expressed in V). Suppose that $\mathcal{T}$ holds in r but m does not hold in r. Then r has a pair p of tuples for which $\mathcal{T}$ holds in p and m does not hold in p.*

Lemma 12 *Let $\mathfrak{F}$(MVD$_\mathcal{R}$) $= (E_\mathcal{R}, \mathcal{L}_\mathsf{M}, \mu_\mathcal{R})$ and $\mathfrak{F}$(MVDF$_\mathcal{I}$) $= (E_\mathcal{I}, \mathcal{L}_\mathsf{M}, \mu_\mathcal{I})$ be, respectively, the frameworks for learning MVD from relations and learning MVDF from interpretations. Let $\mathcal{T} \in \mathcal{L}_\mathsf{M}$ be the target and $\mathcal{H} \in \mathcal{L}_\mathsf{M}$ be the hypothesis. If $r \in \mu_\mathcal{R}(\mathcal{T}) \oplus \mu_\mathcal{R}(\mathcal{H})$ then one can construct in polynomial time in $|\mathcal{T}|$ and $|r|$ an interpretation $\mathcal{I}$ such that $\mathcal{I} \in \mu_\mathcal{I}(\mathcal{T}) \oplus \mu_\mathcal{I}(\mathcal{H})$.*

Proof. Assume that $r \in \mu_\mathcal{R}(\mathcal{T}) \oplus \mu_\mathcal{R}(\mathcal{H})$ is a positive counterexample (the case when r is a negative counterexample is analogous). If $r \notin \mu_\mathcal{R}(\mathcal{H})$ then there is $m \in \mathcal{H}$ such that m does not hold in r. By Lemma 11, r has a pair p of tuples for which $\mathcal{T}$ holds in p and m does not hold in p. Then, $p \in \mu_\mathcal{R}(\mathcal{T}) \setminus \mu_\mathcal{R}(\mathcal{H})$. One can find $p \subseteq r$, by simply checking, for all possible pairs p of tuples in r, whether $\mathcal{H}$ does not hold in p and (with membership queries) whether $\mathcal{T}$ holds in p. Once $p = \{t, t'\}$ is computed, we define $\mathcal{I}$ such that $\mathsf{true}(\mathcal{I}) = \{\gamma \in V \mid t[\gamma] = t'[\gamma]\}$. By definition of $\mathcal{I}$, we have that, for any $m' \in \mathcal{T} \cup \mathcal{H}$, m' does not hold in p if, and only if, $\mathcal{I}$ violates m'. Then, $\mathcal{I} \in \mu_\mathcal{I}(\mathcal{T}) \oplus \mu_\mathcal{I}(\mathcal{H})$. ❑

Lemma 10 shows how one can compute f_MEM (described in Definition 23) with $p_2(|\mathcal{T}|, |\mathcal{I}|) = k \cdot |\mathcal{I}|$ steps, for some constant k. Lemma 12 shows how one can compute f_EQ in $p_3(|\mathcal{T}|, |r|) = k \cdot |r|^2$, for some constant k. Also, we have seen in Section 3 that the size of the hypothesis $\mathcal{H}$ computed by Algorithm 1 is bounded by $|V| \cdot |\mathcal{T}|$. Then, $p_1(|\mathcal{T}|) = |V| \cdot |\mathcal{T}|$. Using Theorems 8 and 9 we can state the following.

Theorem 13 *The problem of learning MVD from relations, more precisely, the learning framework $\mathfrak{F}$(MVD$_\mathcal{R}$), is polynomial time exactly learnable.*

5 Discussion

We solved the open question raised by [22], showing a polynomial time algorithm that exactly learns the class MVDF from interpretations. From a database design perspective, a transformation of our algorithm can be used to extract multivalued dependencies from examples of relations. This process is a sort of knowledge discovery, which can help in restructuring databases and finding data dependencies that database designers did not foresee. From a theoretical point of view, we take a step towards identifying important concept classes that can be learned in polynomial time, a natural research topic in computational learning theory. However, it remains open the question of whether the class MVDF can be exactly learned in polynomial time from entailments (where the entailments are mvd clauses). We know that, for propositional Horn, learning from entailments reduces to learning from interpretations. However, for MVDF a similar reduction is not so easy. The main obstacle is the transformation of membership queries, where one needs to decide whether an interpretation is a model of the target using polynomially many entailment queries.

References

A Proofs for Section 3

We provide the proofs for the lemmas stated in Section 3. We note that our algorithm maintains a sequence of positive examples, as in [7]. Also, the construction of mvd clauses in the hypothesis is inspired by [23].

Remark: In our proof we only consider interpretations $\mathcal{I}$ such that $|\mathsf{false}(\mathcal{I})| \geq 2$. This is justified by the fact that in Line 2 of Algorithm 1 we check whether $\mathcal{T} \models V \rightarrow \mathbf{F}$ and whether $\mathcal{T} \models V \setminus \{v\} \rightarrow v$, for all $v \in V$, and if so we add them to $\mathcal{H}_0$ (note that this can be easily checked with queries to $\mathsf{MEM}_{\mathcal{T},\mathfrak{F}(\text{MVDF}_{\mathcal{I}})}$). Any negative counterexample $\mathcal{I}$ received by Algorithm 1 is such that $|\mathsf{false}(\mathcal{I})| \geq 2$ and it can only violate mvd clauses $X \rightarrow Y \vee Z \in \mathcal{T}$ with Y and Z non-empty. Also, any positive counterexample can only violate mvd clauses $X \rightarrow Y \vee Z \in \mathcal{H}$ with Y and Z non-empty. We consistently use $\mathfrak{P}$ and $\mathfrak{L}$ for the sequences of positive and negative examples of Algorithm 1, respectively. Before we show Lemma 2 we need the following technical lemma.

Lemma 14 *Let $\mathcal{I}$ be a negative example for $\mathcal{T}$ that covers $X \rightarrow Y \vee Z \in \mathcal{T}$. Let* BUILDCLAUSES*($\mathcal{I}, \mathfrak{P}$) be the set* $\{\mathsf{true}(\mathcal{I}) \rightarrow Y_1 \vee Z_1, \ldots, \mathsf{true}(\mathcal{I}) \rightarrow Y_k \vee Z_k\}$. *Then, for all i, $1 \leq i \leq k$, either $Y_i \subseteq Y$ or $Y_i \subseteq Z$.*

Proof. The proof is by induction on the number of elements in $\mathfrak{P}$. The lemma is true when $\mathfrak{P}$ is empty because Function 'BuildClauses' (Algorithm 3) returns the set constructed in Line 1, which contains an mvd clause $\mathsf{true}(\mathcal{I}) \rightarrow v \vee V \setminus (\mathsf{true}(\mathcal{I}) \cup \{v\})$ for each $v \in \mathsf{false}(\mathcal{I})$. Now suppose that the lemma holds for $\mathfrak{P}$ with $m \in \mathbb{N}$ elements. We show that it holds for $\mathfrak{P}$ with $m+1$ elements. Let $\mathcal{P}$ be a fresh positive example (for $\mathcal{T}$). If $\mathcal{P} \models$ BUILDCLAUSES($\mathcal{I}, \mathfrak{P}$) then BUILDCLAUSES($\mathcal{I}, \mathfrak{P}$) $=$ BUILDCLAUSES($\mathcal{I}, \mathfrak{P} \cdot \mathcal{P}$). So, by induction hypothesis the lemma holds.

Otherwise, $\mathcal{P} \not\models$ BUILDCLAUSES($\mathcal{I}, \mathfrak{P}$). Let $\mathsf{true}(\mathcal{I}) \rightarrow Y_1 \vee Z_1, \ldots, \mathsf{true}(\mathcal{I}) \rightarrow Y_k \vee Z_k$ be the mvd clauses in BUILDCLAUSES($\mathcal{I}, \mathfrak{P}$) violated by $\mathcal{P}$. These mvd clauses are replaced, in BUILDCLAUSES($\mathcal{I}, \mathfrak{P} \cdot \mathcal{P}$), by $\mathsf{true}(\mathcal{I}) \rightarrow \bigcup_{j=1}^{k} Y_j \vee (\bigcup_{j=1}^{k} Z_j \setminus \bigcup_{j=1}^{k} Y_j)$. So we need to show that either $\bigcup_{j=1}^{k} Y_j \subseteq Y$ or $\bigcup_{j=1}^{k} Y_j \subseteq Z$. As $\mathcal{P}$ violates these mvd clauses, we have that $\mathsf{true}(\mathcal{I}) \subseteq \mathsf{true}(\mathcal{P})$ and $\mathcal{P}$ must have some zero in Y_j for all $1 \leq j \leq k$. Also, since $\mathcal{P}$ is a positive example and $X \subseteq \mathsf{true}(\mathcal{I})$ either $\mathsf{false}(\mathcal{P}) \subseteq Y$ or $\mathsf{false}(\mathcal{P}) \subseteq Z$. Therefore, either (a) each Y_j has at least one variable in Y or (b) each Y_j has at least one variable in Z. In case (a), by induction hypothesis, either $Y_j \subseteq Y$ or $Y_j \subseteq Z$. As $Y \cap Z = \emptyset$, $Y_j \subseteq Y$ for all $1 \leq j \leq k$. Therefore $\bigcup_{j=1}^{k} Y_j \subseteq Y$. One can prove in the same way that in case (b) we have $\bigcup_{j=1}^{k} Y_j \subseteq Z$. ❑

Lemma 2 (restated). *Let $\mathcal{I}$ be a negative example. Let* BUILDCLAUSES*$(\mathcal{I}, \mathfrak{P})$ $= \{\mathsf{true}(\mathcal{I}) \to Y_1 \vee Z_1, \ldots, \mathsf{true}(\mathcal{I}) \to Y_k \vee Z_k\}$. Then, for all i, j, such that $1 \leq i < j \leq k$, we have that $(*)$: $Y_i \cap Y_j = \emptyset$ and $\bigcup_{j=1}^{k} Y_j = \mathsf{false}(\mathcal{I})$. Moreover, for any $\mathsf{true}(\mathcal{I}) \to Y_i \vee Z_i$, $1 \leq i \leq k$, we have that Y_i, Z_i are non-empty.*

Proof. The proof is by induction on the size of $\mathfrak{P}$. The lemma is true when $\mathfrak{P}$ is empty because Function 'BuildClauses' (Algorithm 3) returns the set constructed in Line 1, which contains an mvd clause $X \to v \vee V \setminus (X \cup \{v\})$ for each $v \in \mathsf{false}(\mathcal{I})$, where $X = \mathsf{true}(\mathcal{I})$ (note that, as in Remark A, $|\mathsf{false}(\mathcal{I})| \geq 2$ and therefore $V \setminus (X \cup \{v\})$ is non-empty). Now suppose that the lemma holds for $\mathfrak{P}$ with $m \in \mathbb{N}$ elements. We show that it holds for $\mathfrak{P}$ with $m+1$ elements. Let $\mathcal{P}$ be a fresh positive example. If $\mathcal{P} \models$ BUILDCLAUSES$(\mathcal{I}, \mathfrak{P})$ then BUILDCLAUSES$(\mathcal{I}, \mathfrak{P}) =$ BUILDCLAUSES$(\mathcal{I}, \mathfrak{P} \cdot \mathcal{P})$. So, by induction hypothesis the lemma holds. Otherwise, $\mathcal{P} \not\models$ BUILDCLAUSES$(\mathcal{I}, \mathfrak{P})$. Let $X \to Y_1 \vee Z_1, \ldots, X \to Y_k \vee Z_k$ be the mvd clauses in BUILDCLAUSES$(\mathcal{I}, \mathfrak{P})$ violated by $\mathcal{P}$. These mvd clauses are replaced, in BUILDCLAUSES$(\mathcal{I}, \mathfrak{P} \cdot \mathcal{P})$, by $X \to \bigcup_{j=1}^{k} Y_j \vee (\bigcup_{j=1}^{k} Z_j \setminus \bigcup_{j=1}^{k} Y_j)$. Clearly, $(*)$ holds in BUILDCLAUSES$(\mathcal{I}, \mathfrak{P} \cdot \mathcal{P})$. It remains to show that $(\bigcup_{j=1}^{k} Z_j \setminus \bigcup_{j=1}^{k} Y_j)$ is not empty. Since $\mathcal{I}$ is a negative example, it violates some clause $X' \to Y' \vee Z' \in \mathcal{T}$ with Y', Z' non-empty (see Remark A). Now suppose to the contrary that $(\bigcup_{j=1}^{k} Z_j \setminus \bigcup_{j=1}^{k} Y_j)$ is empty. Then, $\bigcup_{j=1}^{k} Y_j = \mathsf{false}(\mathcal{I})$ and, by Lemma 14, $\bigcup_{j=1}^{k} Y_j$ is included either in Y' or in Z'. If $\mathsf{false}(\mathcal{I})$ is included either in Y' or in Z' then $\mathcal{I}$ does not violate $X' \to Y' \vee Z'$. This contradicts our assumption that $\mathcal{I}$ violates $X' \to Y' \vee Z' \in \mathcal{T}$. ❑

We now want to show Lemma 4. Before we prove Lemma 4, we need Lemmas 15-20 below.

Lemma 15 *Assume that an interpretation $\mathcal{I}$ violates $X \to Y \vee Z \in \mathcal{T}$. For all $\mathcal{I}_i \in \mathfrak{L}$ such that $\mathcal{I}_i$ covers $X \to Y \vee Z$, $\mathsf{true}(\mathcal{I}_i) \subseteq \mathsf{true}(\mathcal{I})$ if, and only if, $\mathcal{I} \not\models$* BUILDCLAUSES*$(\mathcal{I}_i, \mathfrak{P})$.*

Proof. The $(\Leftarrow)$ direction is trivial. Now, suppose that $\mathsf{true}(\mathcal{I}_i) \subseteq \mathsf{true}(\mathcal{I})$ to prove $(\Rightarrow)$. As $\mathcal{I} \not\models X \to Y \vee Z$, we have that $X \subseteq \mathsf{true}(\mathcal{I})$ and there are $v \in Y$ and $w \in Z$ such that $v, w \in \mathsf{false}(\mathcal{I})$. As $\mathsf{true}(\mathcal{I}_i) \subseteq \mathsf{true}(\mathcal{I})$, we have that $v, w \in \mathsf{false}(\mathcal{I}_i)$. By Lemma 2, there are $\mathsf{true}(\mathcal{I}_i) \to Y_1 \vee Z_1, \mathsf{true}(\mathcal{I}_i) \to Y_2 \vee Z_2 \in$ BUILDCLAUSES$(\mathcal{I}_i, \mathfrak{P})$ such that $v \in Y_1$ and $w \in Y_2$. By Lemma 14, $Y_1 \subseteq Y$ and $Y_2 \subseteq Z$. As $Y \cap Z = \emptyset$, we have that $Y_1 \cap Y_2 = \emptyset$. So, $v \in Z_2$ and $w \in Z_1$, which means that $\mathcal{I}$ violates both $\mathsf{true}(\mathcal{I}_i) \to Y_1 \vee Z_1$ and $\mathsf{true}(\mathcal{I}_i) \to Y_2 \vee Z_2$ in BUILDCLAUSES$(\mathcal{I}_i, \mathfrak{P})$. ❑

We can see the hypothesis $\mathcal{H}$ as a sequence of sets of multivalued clauses, where each $\mathcal{H}_i$ corresponds to the output of Function 'BuildClauses' (Algorithm 3) with $\mathcal{I}_i \in \mathfrak{L}$ and $\mathfrak{P}$ as input.

Lemma 16 *At the end of each iteration, $\mathcal{I}_i \models \mathcal{H} \setminus \mathcal{H}_i$, for all $\mathcal{I}_i \in \mathfrak{L}$.*

Proof. Let $\mathcal{J}$ be the interpretation computed in Line 9 of Algorithm 1. If Algorithm 1 executes Line 16 then it holds that $\mathcal{J} \models \mathcal{H}$. If there is $\mathcal{I}_j \in \mathfrak{L}$ such that $\mathcal{I}_j \not\models$BUILDCLAUSES($\mathcal{J}, \mathfrak{P}$) then $\mathsf{true}(\mathcal{J}) \subset \mathsf{true}(\mathcal{I}_j)$ and the pair $(\mathcal{I}_j, \mathcal{J})$ is a $\mathsf{goodCandidate}$. This contradicts the fact that Algorithm 1 did not replace some interpretation in $\mathfrak{L}$. Otherwise, Algorithm 1 executes Lines 12 and 13, replacing an interpretation $\mathcal{I}_i \in \mathfrak{L}$ by $\mathcal{J}$, where the pair $(\mathcal{I}_i, \mathcal{J})$ is a $\mathsf{goodCandidate}$. In this case, by Definition 1 part (ii), $\mathcal{I}_i \cap \mathcal{J} \models \mathcal{H}$. It remains to check that for any other $\mathcal{I}_j \in \mathfrak{L}$ it holds that $\mathcal{I}_j \models$BUILDCLAUSES($\mathcal{J}, \mathfrak{P}$), but this is always true because of Line 14. ❑

We also require the following technical lemma from [16].

Lemma 17 ([16]) *Let $\mathcal{T}$ be a set of mvd clauses. If $\mathcal{I}$ and $\mathcal{J}$ are interpretations such that $\mathcal{I} \models \mathcal{T}$ and $\mathcal{J} \models \mathcal{T}$, but $\mathcal{I} \cap \mathcal{J} \not\models \mathcal{T}$, then true$(\mathcal{I}) \cup$ true$(\mathcal{J}) = V$.*

Lemma 18 *If Algorithm 1 replaces some $\mathcal{I}_i \in \mathfrak{L}$ with $\mathcal{J}$ then* $\mathsf{false}(\mathcal{I}_i) \subseteq \mathsf{false}(\mathcal{J})$ *($\mathcal{I}_i$ before the replacement).*

Proof. Suppose to the contrary that $\mathsf{false}(\mathcal{I}_i) \not\subseteq \mathsf{false}(\mathcal{J})$. That is, $(*)$ $\mathsf{true}(\mathcal{J} \cap \mathcal{I}_i) \subset \mathsf{true}(\mathcal{J})$. If Algorithm 1 replaced $\mathcal{I}_i \in \mathfrak{L}$ by $\mathcal{J}$ then $(\mathcal{I}_i, \mathcal{J})$ is a $\mathsf{goodCandidate}$. Then, $\mathcal{I}_i \cap \mathcal{J} \not\models \mathcal{T}$ and $\mathcal{I}_i \cap \mathcal{J} \models \mathcal{H}$. If (i) $\mathsf{true}(\mathcal{J} \cap \mathcal{I}_i) \subset \mathsf{true}(\mathcal{J})$ (by $(*)$), (ii) $\mathcal{J} \cap \mathcal{I}_i \models \mathcal{H}$ and (iii) $\mathcal{J} \cap \mathcal{I}_i \not\models \mathcal{T}$; then $(\mathcal{J}, \mathcal{I}_i)$ is a $\mathsf{goodCandidate}$. This contradicts the condition in Line 2 of Algorithm 2, which would not return $\mathcal{J}$ but make a recursive call with $\mathcal{J} \cap \mathcal{I}_i$ and, thus, $\mathsf{false}(\mathcal{I}_i) \subseteq \mathsf{false}(\mathcal{J})$. ❑

Lemma 19 *Let $\mathcal{I}_i, \mathcal{I}_j \in \mathfrak{L}$ and assume $i < j$. At the end of each iteration, if $c \in \mathcal{T}$ is violated by $\mathcal{I}_i, \mathcal{I}_j \in \mathfrak{L}$ then the pair $(\mathcal{I}_i, \mathcal{I}_j)$ is a* $\mathsf{goodCandidate}$ *or* $\mathsf{false}(\mathcal{I}_i) \cap \mathsf{false}(\mathcal{I}_j) = \emptyset$.

Proof. We prove that if $\mathsf{false}(\mathcal{I}_i) \cap \mathsf{false}(\mathcal{I}_j) \neq \emptyset$, then $(\mathcal{I}_i, \mathcal{I}_j)$ is a $\mathsf{goodCandidate}$. By Lemma 15, $\mathsf{true}(\mathcal{I}_i) \subseteq \mathsf{true}(\mathcal{I}_j)$ if, and only if, $\mathcal{I}_i \not\models$ BUILDCLAUSES($\mathcal{I}_j, \mathfrak{P}$). If $\mathcal{I}_i$ covers $c \in \mathcal{T}$ and $\mathcal{I}_j$ violates $c \in \mathcal{T}$ then it follows from Lemma 16 that $\mathsf{true}(\mathcal{I}_i) \not\subseteq \mathsf{true}(\mathcal{I}_j)$. So (i) $\mathsf{true}(\mathcal{I}_i \cap \mathcal{I}_j) \subset \mathsf{true}(\mathcal{I}_i)$. Also by Lemma 16, it holds that $\mathcal{I}_i \models \mathcal{H} \setminus (\mathcal{H}_i \cup \mathcal{H}_j)$ and $\mathcal{I}_j \models \mathcal{H} \setminus (\mathcal{H}_i \cup \mathcal{H}_j)$. Now, by Lemma 17, $\mathsf{false}(\mathcal{I}_j) \cap \mathsf{false}(\mathcal{J}) \neq \emptyset$ implies that $\mathcal{I}_i \cap \mathcal{I}_j \models \mathcal{H} \setminus (\mathcal{H}_i \cup \mathcal{H}_j)$. Since $\mathsf{true}(\mathcal{I}_i \cap \mathcal{I}_j) \subset \mathsf{true}(\mathcal{I}_i)$, we actually have that (ii) $\mathcal{I}_i \cap \mathcal{I}_j \models \mathcal{H}$. To finish, we know that (iii) $\mathcal{I}_i \cap \mathcal{I}_j \not\models \mathcal{T}$ because $c \in \mathcal{T}$ is violated by both $\mathcal{I}_i$ and $\mathcal{I}_j$. Hence, we obtain the conditions (i), (ii), and (iii) of Definition 1, and therefore the pair $(\mathcal{I}_i, \mathcal{I}_j)$ is a $\mathsf{goodCandidate}$. ❑

Lemma 20 *Let $\mathcal{I}_i, \mathcal{I}_j \in \mathfrak{L}$ and assume $i < j$. At the end of each iteration, the pair $(\mathcal{I}_i, \mathcal{I}_j)$ is not a* $\mathsf{goodCandidate}$ *or* $\mathsf{false}(\mathcal{I}_i) \cap \mathsf{false}(\mathcal{I}_j) = \emptyset$.

Proof. Let $\mathcal{J}$ be a countermodel computed in Line 9 of Algorithm 1. Consider the possibilities.

- If Algorithm 1 appends $\mathcal{J}$ to $\mathfrak{L}$, then for all $\mathcal{I}_k \in \mathfrak{L}$ the pair $(\mathcal{I}_k, \mathcal{J})$ cannot be a $\mathsf{goodCandidate}$, because otherwise the condition in Line 10 would be satisfied and, instead of appending $\mathcal{J}$, Algorithm 1 would replace some interpretation $\mathcal{I}_k \in \mathfrak{L}$.
- Now assume that Algorithm 1 replaces (a) $\mathcal{I}_i$ by $\mathcal{J}$ or (b) $\mathcal{I}_j$ by $\mathcal{J}$. Suppose the lemma fails to hold in case (a). The pair $(\mathcal{J}, \mathcal{I}_j)$ is a $\mathsf{goodCandidate}$. This contradicts the condition in Line 2 of Algorithm 2, which would not return $\mathcal{J}$ but make a recursive call with $\mathcal{J} \cap \mathcal{I}_j$. Now, suppose the lemma fails to hold in case (b). The pair $(\mathcal{I}_i, \mathcal{J})$ is a $\mathsf{goodCandidate}$. This contradicts the fact that in Line 11 of Algorithm 1, the first $\mathsf{goodCandidate}$ is replaced and since $i < j$, $\mathcal{I}_i$ should be replaced instead of $\mathcal{I}_j$.
- It remains to check the case where Algorithm 1 replaces $\mathcal{I} \in \mathfrak{L} \setminus \{\mathcal{I}_i, \mathcal{I}_j\}$ by $\mathcal{J}$. We prove that if at the end of the iteration, the pair $(\mathcal{I}_i, \mathcal{I}_j)$ is a $\mathsf{goodCandidate}$ then $\mathsf{false}(\mathcal{I}_i) \cap \mathsf{false}(\mathcal{I}_j) = \emptyset$. So assume that (i) $\mathsf{true}(\mathcal{I}_i \cap \mathcal{I}_j) \subset \mathsf{true}(\mathcal{I}_i)$; (ii) $\mathcal{I}_i \cap \mathcal{I}_j \models \mathcal{H}$; and (iii) $\mathcal{I}_i \cap \mathcal{I}_j \not\models \mathcal{T}$. Point (ii) implies that $\mathcal{I}_i \cap \mathcal{I}_j \models \mathcal{H}_i$ and $\mathcal{I}_i \cap \mathcal{I}_j \models \mathcal{H}_j$. Denote by $\mathcal{H}'$ the hypothesis at the beginning of the iteration. By induction hypothesis, before the replacement, $(\mathcal{I}_i, \mathcal{I}_j)$ was not a $\mathsf{goodCandidate}$ (or $\mathsf{false}(\mathcal{I}_i) \cap \mathsf{false}(\mathcal{I}_j) = \emptyset$ and we are done). Therefore, $\mathcal{I}_i \cap \mathcal{I}_j \not\models \mathcal{H}'$, and there is $\mathcal{H}'_k$ such that $\mathcal{I}_i \cap \mathcal{I}_j \not\models \mathcal{H}'_k$. We know that $k \notin \{i, j\}$ because $\mathcal{H}_i = \mathcal{H}'_i$ and $\mathcal{H}_j = \mathcal{H}'_j$. As $\mathcal{I}_j \models \mathcal{H}' \setminus \mathcal{H}'_j$ (by Lemma 16), we have that $\mathcal{I}_j \models \mathcal{H}'_k$. By the same argument $\mathcal{I}_i \models \mathcal{H}'_k$. Hence, by Lemma 17, $\mathsf{false}(\mathcal{I}_i) \cap \mathsf{false}(\mathcal{I}_j) = \emptyset$.

❑

We are now ready for Lemma 4.

Lemma 4 (restated). *Let $\mathcal{I}_i, \mathcal{I}_j \in \mathfrak{L}$ and assume $i < j$. At the end of each iteration, if $\mathcal{I}_i, \mathcal{I}_j \in \mathfrak{L}$ violate $c \in \mathcal{T}$ then* $\mathsf{false}(\mathcal{I}_i) \cap \mathsf{false}(\mathcal{I}_j) = \emptyset$.

Proof. On one hand, by Lemma 19 the pair $(\mathcal{I}_i, \mathcal{I}_j)$ is a $\mathsf{goodCandidate}$ or $\mathsf{false}(\mathcal{I}_i) \cap \mathsf{false}(\mathcal{I}_j) = \emptyset$. On the other, by Lemma 20 the pair $(\mathcal{I}_i, \mathcal{I}_j)$ is not a $\mathsf{goodCandidate}$ or $\mathsf{false}(\mathcal{I}_i) \cap \mathsf{false}(\mathcal{I}_j) = \emptyset$. We conclude that $\mathsf{false}(\mathcal{I}_i) \cap \mathsf{false}(\mathcal{I}_j) = \emptyset$. ❑

Lemma 6 shows that (1) if an interpretation $\mathcal{I}_i$ is replaced and an element $\mathcal{I}_j$ is removed from $\mathfrak{L}$ then they are mutually disjoint; and (2) if any two elements are removed then they are mutually disjoint. Lemmas 21 and 22 below prepare for the proof of Lemma 6.

Lemma 21 *Let $\mathcal{P}$ and $\mathcal{I}$ be a positive and a negative example, respectively. If $\mathcal{P} \in \mathfrak{P}$ then $\mathcal{P} \models$* BUILDCLAUSES*($\mathcal{I}, \mathfrak{P}$).*

Proof. The proof is by induction on the number of elements in $\mathfrak{P}$. In the base case $\mathfrak{P}$ is empty, so the lemma holds trivially. Now suppose that the lemma holds for $\mathfrak{P}$ with $m \in \mathbb{N}$ elements. We show that it holds for $\mathfrak{P}$ with $m+1$ elements. Let $\mathcal{P}$ be a fresh positive example. We first want to show that $\mathcal{P} \models$ BUILDCLAUSES$(\mathcal{I}, \mathfrak{P} \cdot \mathcal{P})$. If $\mathcal{P} \models$ BUILDCLAUSES$(\mathcal{I}, \mathfrak{P})$ then BUILDCLAUSES$(\mathcal{I}, \mathfrak{P})$ = BUILDCLAUSES$(\mathcal{I}, \mathfrak{P} \cdot \mathcal{P})$. So, by induction hypothesis, the lemma holds.

Otherwise, $\mathcal{P} \not\models$ BUILDCLAUSES$(\mathcal{I}, \mathfrak{P})$. Let $X \to Y_1 \vee Z_1, \ldots, X \to Y_k \vee Z_k$ be the mvd clauses in BUILDCLAUSES$(\mathcal{I}, \mathfrak{P})$ violated by $\mathcal{P}$, where $\mathsf{true}(\mathcal{I}) = X$. These mvd clauses are replaced, in BUILDCLAUSES$(\mathcal{I}, \mathfrak{P} \cdot \mathcal{P})$, by $X \to \bigcup_{j=1}^{k} Y_j \vee (\bigcup_{j=1}^{k} Z_j \setminus \bigcup_{j=1}^{k} Y_j)$. For short denote the latter mvd clause by $X \to Y' \vee Z'$. Suppose to the contrary that $\mathcal{P} \not\models$ BUILDCLAUSES$(\mathcal{I}, \mathfrak{P} \cdot \mathcal{P})$. By construction of BUILDCLAUSES$(\mathcal{I}, \mathfrak{P} \cdot \mathcal{P})$, the only mvd clause that can be violated by $\mathcal{P}$ is $X \to Y' \vee Z'$. Then, there is $v, w \in \mathsf{false}(\mathcal{P})$ such that $v \in Y'$ and $w \in Z'$. By definition of $X \to Y' \vee Z'$, there is $X \to Y_i \vee Z_i \in$ BUILDCLAUSES$(\mathcal{I}, \mathfrak{P})$ such that $w \in Z_i$. If $w \in Z_i$ then, by Lemma 2, there is $X \to Y_j \vee Z_j \in$ BUILDCLAUSES$(\mathcal{I}, \mathfrak{P})$ such that $w \in Y_j$. If $\mathcal{P} \not\models X \to Y_j \vee Z_j$ then this contradicts the fact that $w \in Z'$. Otherwise, $\mathcal{P} \models X \to Y_j \vee Z_j$. So, $\mathsf{false}(\mathcal{P}) \subseteq Y_j$ and $X \to Y_j \vee Z_j \in$ BUILDCLAUSES$(\mathcal{I}, \mathfrak{P})$. As $v \in Y_j \cap Y'$, we have that $Y_j \cap Y' \neq \emptyset$. This contradicts Lemma 2.

It remains to show that for any other $\mathcal{P}' \in \mathfrak{P}$, we have that $\mathcal{P}' \models$ BUILDCLAUSES$(\mathcal{I}, \mathfrak{P}' \cdot \mathcal{P})$. If $\mathcal{P}' \not\models$ BUILDCLAUSES$(\mathcal{I}, \mathfrak{P}' \cdot \mathcal{P})$ then the only clause that can be violated by $\mathcal{P}'$ is $X \to Y' \vee Z'$. Then, there is $v', w' \in \mathsf{false}(\mathcal{P}')$ such that $v' \in Y'$ and $w' \in Z'$. Therefore, $v' \in Y_i$, for some $X \to Y_i \vee Z_i \in$ BUILDCLAUSES$(\mathcal{I}, \mathfrak{P})$ violated by $\mathcal{P}$. If $w' \in Z'$ then, as $Z' = (\bigcup_{j=1}^{k} Z_j \setminus \bigcup_{j=1}^{k} Y_j) = \bigcap_{j=1}^{k} Z_j$, we have that $w' \in Z_i$. Then, $\mathcal{P}' \not\models X \to Y_i \vee Z_i$. This, contradicts the fact that, by induction hypothesis, $\mathcal{P}' \models$ BUILDCLAUSES$(\mathcal{I}, \mathfrak{P})$. ❑

Lemma 22 *Let $\mathcal{I}$, $\mathcal{J}$ and $\mathcal{K}$ be negative examples such that* $\mathsf{true}(\mathcal{I}) \subseteq \mathsf{true}(\mathcal{J}) \subseteq \mathsf{true}(\mathcal{K})$. *If $\mathcal{K} \not\models$* BUILDCLAUSES*$(\mathcal{I}, \mathfrak{P})$ then $\mathcal{K} \not\models$* BUILDCLAUSES*$(\mathcal{J}, \mathfrak{P})$.*

Proof. If $\mathcal{K} \not\models$ BUILDCLAUSES$(\mathcal{I}, \mathfrak{P})$ then there is $\mathsf{true}(\mathcal{I}) \to Y \vee Z \in$ BUILDCLAUSES$(\mathcal{I}, \mathfrak{P})$ with $v \in Y$, $w \in Z$ such that $v, w \in \mathsf{false}(\mathcal{K})$. If $v, w \in \mathsf{false}(\mathcal{K})$ then $v, w \in \mathsf{false}(\mathcal{J})$. If there is $\mathsf{true}(\mathcal{J}) \to Y' \vee Z' \in$ BUILDCLAUSES$(\mathcal{J}, \mathfrak{P})$ with $v \in Y'$, $w \in Z'$ then $\mathcal{K} \not\models$ BUILDCLAUSES$(\mathcal{J}, \mathfrak{P})$. Otherwise, there is no such mvd clause in BUILDCLAUSES$(\mathcal{J}, \mathfrak{P})$. This means that there is $\mathcal{P} \in \mathfrak{P}$ such that $\mathsf{true}(\mathcal{J}) \subseteq \mathsf{true}(\mathcal{P})$ and $v, w \in \mathsf{false}(\mathcal{P})$. As $\mathsf{true}(\mathcal{J}) \subseteq \mathsf{true}(\mathcal{P})$, we have that $\mathsf{true}(\mathcal{I}) \subseteq \mathsf{true}(\mathcal{P})$. Then, $\mathcal{P} \not\models$ BUILDCLAUSES$(\mathcal{I}, \mathfrak{P})$. Since $\mathcal{P} \in \mathfrak{P}$, this contradicts Lemma 21. ❑

We can now prove Lemma 6.

Lemma 6 (restated). *In Line 14 of Algorithm 1, the following holds:*

1. *if $\mathcal{I}_j$ is removed after the replacement of some $\mathcal{I}_i \in \mathfrak{L}$ by $\mathcal{J}$ (Line 13) then* $\mathsf{false}(\mathcal{I}_i) \cap \mathsf{false}(\mathcal{I}_j) = \emptyset$ *($\mathcal{I}_i$ before the replacement);*

2. *if $\mathcal{I}_j, \mathcal{I}_k$ with $j < k$ are removed after the replacement of some $\mathcal{I}_i \in \mathfrak{L}$ by $\mathcal{J}$ (Line 13) then* $\mathsf{false}(\mathcal{I}_j) \cap \mathsf{false}(\mathcal{I}_k) = \emptyset$.

Proof. We first argue that if $\mathcal{I}_j$ is removed then $i < j$. Suppose to the contrary that $j < i$ and $\mathcal{I}_j$ is removed after the replacement of $\mathcal{I}_i$ by $\mathcal{J}$. Then, $\mathcal{I}_j \not\models$BuildClauses($\mathcal{J}$), which means that $\mathsf{true}(\mathcal{J}) \subset \mathsf{true}(\mathcal{I}_j)$. We have that (i) $\mathsf{true}(\mathcal{I}_j \cap \mathcal{J}) \subset \mathsf{true}(\mathcal{I}_j)$; (ii) $\mathcal{I}_j \cap \mathcal{J} \models \mathcal{H}$ and (iii) $(\mathcal{I}_j \cap \mathcal{J}) \not\models \mathcal{T}$ (as $\mathcal{I}_j \cap \mathcal{J} = \mathcal{J}$). Then, by Definition 1, the pair $(\mathcal{I}_j, \mathcal{J})$ is a $\mathsf{goodCandidate}$. This contradicts the fact that in Line 11 of Algorithm 1, the first $\mathsf{goodCandidate}$ is replaced.

So we can assume that $i < j < k$. We now argue that under the conditions stated by this lemma if $\mathsf{false}(\mathcal{I}_i) \cap \mathsf{false}(\mathcal{I}_j) = \emptyset$ (respectively, $\mathsf{false}(\mathcal{I}_j) \cap \mathsf{false}(\mathcal{I}_k) = \emptyset$) does not hold then the pair $(\mathcal{I}_i, \mathcal{I}_j)$ (respectively, $(\mathcal{I}_j, \mathcal{I}_k)$) is a $\mathsf{goodCandidate}$ (Definition 1), which contradicts Lemma 20. In our proof by contradiction, we show that conditions (i), (ii) and (iii) of Definition 1 hold for both $(\mathcal{I}_i, \mathcal{I}_j)$ and $(\mathcal{I}_j, \mathcal{I}_k)$.

- For condition (i): assume to the contrary that $\mathsf{true}(\mathcal{I}_i) \subseteq \mathsf{true}(\mathcal{I}_j)$. By Lemma 18, we know that $\mathsf{true}(\mathcal{J}) \subseteq \mathsf{true}(\mathcal{I}_i)$. As $\mathsf{true}(\mathcal{J}) \subseteq \mathsf{true}(\mathcal{I}_i) \subseteq \mathsf{true}(\mathcal{I}_j)$ and $\mathcal{I}_j \not\models$ BUILDCLAUSES($\mathcal{J}, \mathfrak{P}$), by Lemma 22, we have $\mathcal{I}_j \not\models$ BUILDCLAUSES($\mathcal{I}_i, \mathfrak{P}$), which is a contradiction with Lemma 16. Now, we assume to the contrary that $\mathsf{true}(\mathcal{I}_j) \subseteq \mathsf{true}(\mathcal{I}_k)$.

  As $\mathsf{true}(\mathcal{J}) \subseteq \mathsf{true}(\mathcal{I}_j) \subseteq \mathsf{true}(\mathcal{I}_k)$ and $\mathcal{I}_k \not\models$ BUILDCLAUSES($\mathcal{J}, \mathfrak{P}$), by Lemma 22, we have $\mathcal{I}_k \not\models$BUILDCLAUSES($\mathcal{I}_j, \mathfrak{P}$), which is a contradiction with Lemma 16.

- For condition (ii): as $\mathcal{I}_j \models \mathcal{H} \setminus \mathcal{H}_j$ (Lemma 16) we have $\mathcal{I}_j \models \mathcal{H} \setminus (\mathcal{H}_i \cup \mathcal{H}_j)$. By the same argument $\mathcal{I}_i \models \mathcal{H} \setminus (\mathcal{H}_i \cup \mathcal{H}_j)$. If $\mathsf{false}(\mathcal{I}_i) \cap \mathsf{false}(\mathcal{I}_j) \neq \emptyset$ then, by Lemma 17, $\mathcal{I}_i \cap \mathcal{I}_j \models \mathcal{H} \setminus (\mathcal{H}_i \cup \mathcal{H}_j)$. In fact, by the argument above for condition (i), $\mathsf{true}(\mathcal{I}_i) \not\subseteq \mathsf{true}(\mathcal{I}_j)$. So, we actually have $\mathcal{I}_i \cap \mathcal{I}_j \models \mathcal{H}$. Similarly, as $\mathcal{I}_j \models \mathcal{H} \setminus \mathcal{H}_j$ (Lemma 16) we have $\mathcal{I}_j \models \mathcal{H} \setminus (\mathcal{H}_j \cup \mathcal{H}_k)$. By the same argument $\mathcal{I}_k \models \mathcal{H} \setminus (\mathcal{H}_j \cup \mathcal{H}_k)$. If $\mathsf{false}(\mathcal{I}_j) \cap \mathsf{false}(\mathcal{I}_k) \neq \emptyset$ then, by Lemma 17 and the fact that $\mathsf{true}(\mathcal{I}_j) \not\subseteq \mathsf{true}(\mathcal{I}_k)$, we have $\mathcal{I}_j \cap \mathcal{I}_k \models \mathcal{H}$.

- For condition (iii): suppose to the contrary that $\mathcal{I}_i \cap \mathcal{I}_j \models \mathcal{T}$.

  As $\mathcal{I}_j \not\models$BUILDCLAUSES($\mathcal{J}, \mathfrak{P}$) and (by Lemma 18) $\mathsf{true}(\mathcal{J}) \subseteq \mathsf{true}(\mathcal{I}_i)$, we have that $\mathcal{I}_j \cap \mathcal{I}_i \not\models$BUILDCLAUSES($\mathcal{J}, \mathfrak{P}$). Then, the condition in Line 2 of Algorithm 4 is satisfied. So Algorithm 4 appends $\mathcal{I}_i \cap \mathcal{I}_j$ to $\mathfrak{P}$ and recursively calls Function 'UpdatePositiveExamples' with $\mathcal{J}$, $\mathfrak{P}$ and $\mathfrak{L}$ as input. Then, by Lemma 21, $\mathcal{I}_i \cap \mathcal{I}_j \models$BUILDCLAUSES($\mathcal{J}, \mathfrak{P}$). Then, in Line 14 of Algorithm 1, $\mathcal{I}_j \models$BUILDCLAUSES($\mathcal{J}, \mathfrak{P}$), which is a contradiction. Similarly, suppose to the contrary that $\mathcal{I}_j \cap \mathcal{I}_k \models \mathcal{T}$. As both $\mathcal{I}_k, \mathcal{I}_j$ do not satisfy BUILDCLAUSES($\mathcal{J}, \mathfrak{P}$), the condition in Line 2 of Algorithm 4 is satisfied. So Algorithm 4 appends $\mathcal{I}_j \cap \mathcal{I}_k$ to $\mathfrak{P}$ and recursively calls Function 'UpdatePositiveExamples' with $\mathcal{J}$, $\mathfrak{P}$ and $\mathfrak{L}$ as input. Then, by Lemma 21, $\mathcal{I}_j \cap \mathcal{I}_k \models$BUILDCLAUSES($\mathcal{J}, \mathfrak{P}$). Hence, when Line 14 of

Algorithm 1 is executed both $\mathcal{I}_j, \mathcal{I}_k$ satisfy BUILDCLAUSES($\mathcal{J}, \mathfrak{P}$), which is a contradiction.

So conditions (i), (ii) and (iii) of Definition 1 hold for $(\mathcal{I}_i, \mathcal{I}_j)$ and $(\mathcal{I}_j, \mathcal{I}_k)$, which contradicts Lemma 20. Then, $\mathsf{false}(\mathcal{I}_i) \cap \mathsf{false}(\mathcal{I}_j) = \emptyset$ and $\mathsf{false}(\mathcal{I}_j) \cap \mathsf{false}(\mathcal{I}_k) = \emptyset$. ❏

B Proof of Theorem 9

For convenience, we restate our definition of reduction presented in Section 4.

Definition 23 *A learning framework* $\mathfrak{F} = (E, \mathcal{L}, \mu)$ polynomial time reduces *to* $\mathfrak{F}' = (E', \mathcal{L}, \mu')$ *if, for some polynomials* $p_1(\cdot)$*,* $p_2(\cdot,\cdot)$ *and* $p_3(\cdot,\cdot)$ *there exist a function* $f_{\mathsf{MEM}} : \mathcal{L} \times E' \to \{$ *'yes', 'no'*$\}$ *and a partial function* $f_{\mathsf{EQ}} : \mathcal{L} \times \mathcal{L} \times E \to E'$*, defined for every* (l, h, e) *such that* $|h| \leq p_1(|l|)$*, for which the following conditions hold:*

- *for all* $e' \in E'$ *we have* $e' \in \mu'(l)$ *iff* $f_{\mathsf{MEM}}(l, e') =$ *'yes';*
- *for all* $e \in E$ *we have* $e \in \mu(l) \oplus \mu(h)$ *iff* $f_{\mathsf{EQ}}(l, h, e) \in \mu'(l) \oplus \mu'(h)$*;*
- $f_{\mathsf{MEM}}(l, e')$ *and* $f_{\mathsf{EQ}}(l, h, e)$ *are computable in time* $p_2(|l|, |e'|)$ *and* $p_3(|l|, |e|)$*, respectively, and* l *can only be accessed by calls to the membership oracle* $\mathsf{MEM}_{l,\mathfrak{F}}$.

Theorem 9 (restated). *Let* $\mathfrak{F} = (E, \mathcal{L}, \mu)$ *and* $\mathfrak{F}' = (E', \mathcal{L}, \mu')$ *be learning frameworks. If there exists a polynomial time reduction from* $\mathfrak{F}$ *to* $\mathfrak{F}'$ *and* $\mathfrak{F}'$ *is polynomial time exactly learnable then* $\mathfrak{F}$ *is polynomial time exactly learnable.*

Proof. Let A' be a polynomial time learning algorithm for $(E', \mathcal{L}, \mu')$. We construct a learning algorithm A for $(E, \mathcal{L}, \mu)$, using internally the learning algorithm A', as follows. As learning $(E, \mathcal{L}, \mu)$ polynomial time reduces to learning $(E', \mathcal{L}, \mu')$, we have that:

- there are functions $f_{\mathsf{MEM}} : \mathcal{L} \times E' \to \{$ 'yes', 'no'$\}$ and $f_{\mathsf{EQ}} : \mathcal{L} \times \mathcal{L} \times E \to E'$ such that f_{MEM} maps $l \in \mathcal{L}$ and '$e' \in E'$' into 'yes' or 'no' (depending on whether $e' \in \mu'(l)$); and f_{EQ} transforms a counterexample '$e \in E$' into a counterexample '$e' \in E'$'.

So, whenever a membership query with $e' \in E'$ as input is called by A' we compute $f_{\mathsf{MEM}}(l, e')$ by making calls to the $\mathsf{MEM}_{l,\mathfrak{F}}$ oracle. We return 'yes' to A' if $f_{\mathsf{MEM}}(l, e') =$ 'yes' and 'no' otherwise. Whenever an equivalence query with $h \in \mathcal{L}$ as input is called by A' we pass it on to the $\mathsf{EQ}_{l,\mathfrak{F}}$ oracle. If it returns 'yes' then the learner succeeded. Otherwise the oracle returns 'no' and provides a counterexample $e \in E$. Then, we compute $e' = g(l, h, e)$ and return it to A'. Notice that computing $f_{\mathsf{EQ}}(l, h, e)$ may also require posing additional membership queries (recall that l can only be accessed via queries to the oracle $\mathsf{MEM}_{l,\mathfrak{F}}$).

By definition of f_{MEM} and f_{EQ}, all the answers provided to A' are consistent with answers the oracles $\mathsf{MEM}_{l,\mathfrak{F}'}$ and $\mathsf{EQ}_{l,\mathfrak{F}'}$ would provide to A'. Clearly, if algorithm A terminates then it learns l.

It remains to prove the polynomial time bound for A. Let $p_1(\cdot)$, $p_2(\cdot)$ and $p_3(\cdot,\cdot)$ be the polynomials of Definition 23, that is,

- $p_1(|l|)$ is the polynomial bound on $|h|$;
- $p_2(|l|,|e'|)$ is the polynomial time bound for computing $f_{\mathsf{MEM}}(l,e')$;
- $p_3(|l|,|e|)$ is the polynomial time bound for computing $f_{\mathsf{EQ}}(l,h,e)$.

Let $p(\cdot,\cdot)$ be a polynomial such that in every run of A', the time used by A' up to each step of computation is bounded by $p(|l|,|y'|)$, where $|l|$ is the size of the target $l \in \mathcal{L}$ and $|y'|$ is the size of the largest counterexample $y' \in E'$ seen by A' up to that point of computation. As y' is the result of transforming with function f_{EQ} some counterexample $y \in E$ given by the $\mathsf{EQ}_{l,\mathfrak{F}}$ oracle to algorithm A, its size $|y'|$ is bounded by $p_3(|l|,|y|)$. Notice that y is also the largest counterexample seen so far by A. Thus, at every step of computation the time used by A' up to that step is bounded by a polynomial $p'(|l|,|y|) = p(|l|, p_3(|l|,|y|))$.

For every membership query with $e' \in E'$ asked by A', the size of e' does not exceed the polynomial time bound of A' up to that point, that is, $|e'| \leq p'(|l|,|y|)$. Then, the time needed to transform membership queries and answers to equivalence queries is bounded by $p_2'(|l|,|y|) = p_2(|l|, p'(|l|,|y|))$ and $p_3(|l|,|y|)$, respectively. All in all, at every step of computation the time used by A up to that step is bounded by $p'(|l|,|y|) \cdot (p_2'(|l|,|y|) + p_3(|l|,|y|))$, which is polynomial in $|l|$ and $|y|$, as required. ❏

C Reductions among Learning Problems

We now explain the reducibility of the learning problems presented in Figure 1. For convenience, in Figure 2, we enumerate the reductions [4]. Points (1) and (6) follow from the fact that one can express any Horn formula with a polynomial size MVDF (see Remark 2 below). Point (2) is given in Subsection C.2. We then have Point (3), where have the MVD$_{\mathcal{R}}$ $\rightarrow$ MVDF$_{\mathcal{I}}$ direction proved in Subsection 4.1 (note that this also gives Point (8)). The other direction, MVDF$_{\mathcal{I}}$ $\rightarrow$ MVD$_{\mathcal{R}}$, can be proved with similar arguments. Point (4) follows from the fact that CRFMVF is a restriction of MVDF. We show Point (5) in Subsection C.1. Finally, we show Point (7) in Subsection C.3.

We write $\mathsf{ant}(c)$ (the antecedent) for the set of variables that occur negated in a clause c (this set contains $\mathbf{T}$ if no variable occurs negated).

[4]Note that our reduction in Point (1) of Figure 2 is non-proper. Though, in this case one can avoid this by translating the hypothesis to Horn whenever the algorithm poses an equivalence query (see Remark 2).

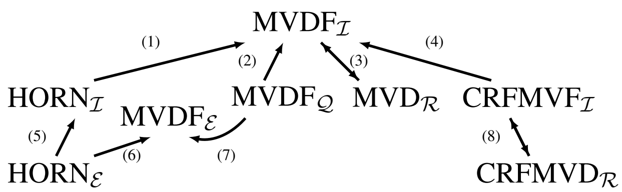

Figure 2: Reductions among learning problems

C.1 Propositional Horn: from Entailments to Interpretations

The learning framework $\mathfrak{F}$(HORN$_{\mathcal{I}}$), studied by [5], is defined as $(E_{\mathcal{I}}, \mathcal{L}_{\mathsf{H}}, \mu_{\mathcal{I}})$, where $\mathcal{L}_{\mathsf{H}}$ is the set of all Horn sentences which can be formulated in a set of variables V, $E_{\mathcal{I}}$ is the set of interpretations over variables in V and, for a Horn sentence $\mathcal{T} \in \mathcal{L}_{\mathsf{H}}$, $\mu_{\mathcal{I}}(\mathcal{T})$ is defined as $\{\mathcal{I} \in E_{\mathcal{I}} \mid \mathcal{I} \models \mathcal{T}\}$. We also define the learning framework $\mathfrak{F}$(HORN$_{\mathcal{E}}$), studied by [13], as $(E_{\mathcal{E}}, \mathcal{L}_{\mathsf{H}}, \mu_{\mathcal{E}})$, where $\mathcal{L}_{\mathsf{H}}$ is the set of all Horn sentences which can be formulated in a set of variables V, $E_{\mathcal{E}}$ is the set of all Horn clauses over variables in V and, for a Horn sentence $\mathcal{T} \in \mathcal{L}_{\mathsf{H}}$, $\mu_{\mathcal{E}}(\mathcal{T})$ is defined as $\{c \in E_{\mathcal{E}} \mid \mathcal{T} \models c\}$.

An algorithm to learn Horn sentences from entailments is presented by [13], where the authors mention that their solution is in fact an application of the learning from interpretations algorithm presented by [5] with some twists. Here we give an alternative proof, based on Theorem 9, which shows that learning Horn sentences from entailments can be reduced in polynomial time to learning Horn sentences from interpretations. To give our proof by reduction we use Angluin's [5] algorithm as a 'black box' and: (1) transform counterexamples given by equivalence queries, which come as entailments into interpretations; and (2) transform the membership queries, which come as interpretations into entailments. Let $\mathcal{T}$ be the target Horn sentence and $\mathcal{H}$ the learner's hypothesis. The following lemma shows how one can simulate an equivalence query by transforming a counterexample in the learning from entailments scenario into a counterexample in the learning from interpretations scenario.

Lemma 24 *Let $\mathfrak{F}$(HORN$_{\mathcal{E}}$) $= (E_{\mathcal{E}}, \mathcal{L}_{\mathsf{H}}, \mu_{\mathcal{E}})$ be the learning Horn from entailments framework and $\mathfrak{F}$(HORN$_{\mathcal{I}}$) $= (E_{\mathcal{I}}, \mathcal{L}_{\mathsf{H}}, \mu_{\mathcal{I}})$ be the learning Horn from interpretations framework. Assume that the target $\mathcal{T}$ and the hypothesis $\mathcal{H}$ are in variables V and $|\mathcal{H}|$ is polynomial in $|\mathcal{T}|$. If $c \in \mu_{\mathcal{E}}(\mathcal{T}) \oplus \mu_{\mathcal{E}}(\mathcal{H})$ then one can construct in time polynomial in $|\mathcal{T}|$ an interpretation $\mathcal{I}$ such that $\mathcal{I} \in \mu_{\mathcal{I}}(\mathcal{T}) \oplus \mu_{\mathcal{I}}(\mathcal{H})$.*

Proof. We show how one can transform a Horn clause c that is a *positive* counterexample (in $\mathfrak{F}$(HORN$_{\mathcal{E}}$)) into a *negative* counterexample (in $\mathfrak{F}$(HORN$_{\mathcal{I}}$)) and vice-versa. If $\mathcal{T} \not\models c$ and $\mathcal{H} \models c$ then we construct an interpretation $\mathcal{I}$ as the result of initially setting $\mathsf{true}(\mathcal{I}) = \mathsf{ant}(c)$ and then exhaustively applying the following rule:

- if $\mathcal{T} \models \bigwedge_{v \in \mathsf{true}(\mathcal{I})} \to w$ (checked with membership query to $\mathsf{MEM}_{\mathcal{T},\mathfrak{F}(\mathrm{HORN}_{\mathcal{E}})}$), where $w \in V \setminus \mathsf{true}(\mathcal{I})$, then add w to $\mathsf{true}(\mathcal{I})$.

The resulting $\mathcal{I}$ is model of $\mathcal{T}$. As $\mathcal{T} \not\models c$ we know that the consequent of c is not in $\mathsf{true}(\mathcal{I})$. Then, since $\mathsf{ant}(c) \subseteq \mathsf{true}(\mathcal{I})$, we have that $\mathcal{I}$ does not satisfy $\mathcal{H}$. That is, $\mathcal{I} \in \mu_{\mathcal{I}}(\mathcal{T}) \oplus \mu_{\mathcal{I}}(\mathcal{H})$. Notice that in this case we made $|V|$ membership queries to the oracle $\mathsf{MEM}_{\mathcal{T},\mathfrak{F}(\mathrm{HORN}_{\mathcal{E}})}$. When $\mathcal{T} \models c$ and $\mathcal{H} \not\models c$ the argument is similar but we need to check whether $\mathcal{H} \models \bigwedge_{v \in \mathsf{true}(\mathcal{I})} \to w$, where $w \in V \setminus \mathsf{true}(\mathcal{I})$. Since in this case we evaluate the hypothesis, no membership query is necessary to produce a negative counterexample. ❑

To simulate membership queries we transform an interpretation $\mathcal{I}$ into polynomially many entailment queries which together decide whether $\mathcal{I}$ satisfies $\mathcal{T}$ or not.

Lemma 25 *Let $\mathfrak{F}$(HORN$_{\mathcal{E}}$) $= (E_{\mathcal{E}}, \mathcal{L}_{\mathsf{H}}, \mu_{\mathcal{E}})$ be the learning Horn from entailments framework and $\mathfrak{F}$(HORN$_{\mathcal{I}}$) $= (E_{\mathcal{I}}, \mathcal{L}_{\mathsf{H}}, \mu_{\mathcal{I}})$ be the learning Horn from interpretations framework. For any interpretation $\mathcal{I}$ of a target concept representation $\mathcal{T} \in \mathcal{L}_{\mathsf{H}}$, one can decide in polynomial time in $|\mathcal{T}|$ whether $\mathcal{I} \in \mu_{\mathcal{I}}(\mathcal{T})$.*

Proof. A very straightforward algorithm to decide whether $\mathcal{I}$ satisfies $\mathcal{T}$ is described as follows. Let $C = \{\bigwedge_{v \in \mathsf{true}(\mathcal{I})} \to z \mid z \in \mathsf{false}(\mathcal{I})\}$. For every $c \in C$ the algorithm calls $\mathsf{MEM}_{\mathcal{T},\mathfrak{F}(\mathrm{HORN}_{\mathcal{E}})}$ asking whether $\mathcal{T} \models c$. If the answer to any of these queries is 'yes' then return 'no'. That is, $\mathcal{I}$ does not satisfy $\mathcal{T}$. Otherwise, return 'yes', $\mathcal{I}$ satisfies $\mathcal{T}$. ❑

Lemmas 24 and 25 show how one can compute, respectively, f_{EQ} and f_{MEM} described in Definition 23. Then, using Theorem 9, we obtain an alternative proof for the result presented by [13].

Theorem 26 ([13]) *The problem of learning propositional Horn from entailments, more precisely, the learning framework $\mathfrak{F}$(HORN$_{\mathcal{E}}$), is polynomial time exactly learnable.*

C.2 Multivalued Dependency Formulas: from 2-quasi-Horn to Interpretations

The learning framework $\mathfrak{F}$(MVDF$_{\mathcal{Q}}$), studied by the authors of [16], is formally defined as $(E_{\mathcal{Q}}, \mathcal{L}_{\mathsf{M}}, \mu_{\mathcal{Q}})$, where $\mathcal{L}_{\mathsf{M}}$ is the set of all MVDFs which can be formulated in a set of variables V, $E_{\mathcal{Q}}$ is the set of 2-quasi-Horn clauses over variables in V and, for a MVDF $\mathcal{T} \in \mathcal{L}_{\mathsf{M}}$, $\mu_{\mathcal{Q}}(\mathcal{T})$ is defined as $\{e \in E_{\mathcal{Q}} \mid \mathcal{T} \models e\}$.

We show that learning MVDF from 2-quasi-Horn clauses is reducible to learning MVDF from interpretations. More precisely, $\mathfrak{F}$(MVDF$_{\mathcal{Q}}$) polynomial time reduces to $\mathfrak{F}$(MVDF$_{\mathcal{I}}$). To give our proof by reduction we use the algorithm presented in Section 3 as a 'black box' and: (1) transform the membership queries, which come as interpretations into 2-quasi-Horn clauses; and (2) transform counterexamples given by equivalence queries, which come as 2-quasi-Horn clauses into interpretations. Let $\mathcal{T}$ be the target MVDF and $\mathcal{H}$ the learner's hypothesis. To simulate membership queries we transform an interpretation $\mathcal{I}$ into polynomially many 2-quasi-Horn queries which together decide whether $\mathcal{I}$ satisfies $\mathcal{T}$ or not.

Lemma 27 *Let $\mathfrak{F}$(MVDF$_\mathcal{Q}$) $= (E_\mathcal{Q}, \mathcal{L}_\mathsf{M}, \mu_\mathcal{Q})$ be the learning MVDF from 2-quasi-Horn framework and $\mathfrak{F}$(MVDF$_\mathcal{I}$) $= (E_\mathcal{I}, \mathcal{L}_\mathsf{M}, \mu_\mathcal{I})$ be the learning MVDF from interpretations framework. For any interpretation $\mathcal{I}$ of a target concept representation $\mathcal{T} \in \mathcal{L}_\mathsf{M}$, one can decide in polynomial time in $|\mathcal{T}|$ whether $\mathcal{I} \in \mu_\mathcal{I}(\mathcal{T})$.*

Proof. A very straightforward algorithm to decide whether $\mathcal{I}$ satisfies $\mathcal{T}$ is described as follows. Let $C = \{\bigwedge_{v \in \mathsf{true}(\mathcal{I})} \to w \vee z \mid w, z \in \mathsf{false}(\mathcal{I})\} \cup \{V \to \mathbf{F} \mid \mathsf{true}(\mathcal{I}) = V\}$. For every $c \in C$ the algorithm calls $\mathsf{MEM}_{\mathcal{T},\mathfrak{F}(\mathrm{MVDF}_\mathcal{Q})}$ asking whether $\mathcal{T} \models c$. If the answer to any of these queries is 'yes' then return 'no'. That is, $\mathcal{I}$ does not satisfy $\mathcal{T}$. Otherwise, return 'yes', $\mathcal{I}$ satisfies $\mathcal{T}$. ❑

We note that in the learning framework $\mathfrak{F}$(MVDF$_\mathcal{Q}$) one can use the membership oracle to ensure that at all times $\mathcal{T} \models \mathcal{H}$. Then, we can assume w.l.o.g. that all counterexamples given by the oracle are positive. To transform positive counterexamples, we employ the following result from [16].

Lemma 28 *(Direct Adaptation from [16]) Let $\mathfrak{F}$(MVDF$_\mathcal{Q}$) $= (E_\mathcal{Q}, \mathcal{L}_\mathsf{M}, \mu_\mathcal{Q})$ be the learning MVDF from 2-quasi-Horn framework and $\mathfrak{F}$(MVDF$_\mathcal{I}$) $= (E_\mathcal{I}, \mathcal{L}_\mathsf{M}, \mu_\mathcal{I})$ be the learning MVDF from interpretations framework. Assume that the target $\mathcal{T}$ and the hypothesis $\mathcal{H}$ are in variables V and $|\mathcal{H}|$ is polynomial in $|\mathcal{T}|$. If $c \in \mu_\mathcal{Q}(\mathcal{T}) \oplus \mu_\mathcal{Q}(\mathcal{H})$ is a positive counterexample then one can construct in time polynomial in $|\mathcal{T}|$ an interpretation $\mathcal{I}$ such that $\mathcal{I} \in \mu_\mathcal{I}(\mathcal{T}) \oplus \mu_\mathcal{I}(\mathcal{H})$ is a negative counterexample.*

The proof of Lemma 28 in [16] involves the construction of a polynomial size semantic tree for the hypothesis $\mathcal{H}$. The transformation of negative 2-quasi-Horn counterexamples is also possible. In this case, we would require additional (polynomially many) membership queries to construct a semantic tree. Lemmas 27 and 28 show how one can compute, respectively, f_{MEM} and f_{EQ} described in Definition 23. Then, using Theorem 9, we obtain an alternative proof for the result presented by [16].

Theorem 29 ([16]) *The problem of learning MVDF from 2-quasi-Horn clauses, more precisely, the learning framework $\mathfrak{F}$(MVDF$_\mathcal{Q}$), is polynomial time exactly learnable.*

The difficulty in showing a reduction in the other direction, from $\mathfrak{F}$(MVDF$_\mathcal{I}$) to $\mathfrak{F}$(MVDF$_\mathcal{Q}$), is to decide whether the target entails a 2-quasi-Horn clause using polynomially many membership queries with interpretations as input.

C.3 Multivalued Dependency Formulas: from 2-quasi-Horn to Entailments (mvd clauses)

The learning framework $\mathfrak{F}$(MVDF$_\mathcal{E}$) is defined as $(E_\mathcal{E}, \mathcal{L}_\mathsf{M}, \mu_\mathcal{E})$, where $\mathcal{L}_\mathsf{M}$ is the set of all MVDFs which can be formulated in a set of variables V, $E_\mathcal{E}$ is the set of mvd clauses over variables in V and, for a MVDF $\mathcal{T} \in \mathcal{L}_\mathsf{M}$, $\mu_\mathcal{E}(\mathcal{T})$ is defined as $\{e \in E_\mathcal{E} \mid \mathcal{T} \models e\}$.

We show that learning MVDF from 2-quasi-Horn clauses is reducible to learning MVDF from entailments. More precisely, $\mathfrak{F}$(MVDF$_\mathcal{Q}$) polynomial time reduces to $\mathfrak{F}$(MVDF$_\mathcal{E}$). To reduce the problem we: (1) transform the membership queries, which come as mvd clauses into 2-quasi-Horn clauses; and (2) transform counterexamples given by equivalence queries, which come as 2-quasi-Horn clauses into mvd clauses. Let $\mathcal{T}$ be the target MVDF and $\mathcal{H}$ the learner's hypothesis. The next lemma is immediate, it follows from the fact that any mvd clause is equivalent to polynomially many 2-quasi-Horn clauses (see Remark 2).

Lemma 30 *Let $\mathfrak{F}$(MVDF$_\mathcal{Q}$) $= (E_\mathcal{Q}, \mathcal{L}_\mathsf{M}, \mu_\mathcal{Q})$ be the learning MVDF from 2-quasi-Horn framework and $\mathfrak{F}$(MVDF$_\mathcal{E}$) $= (E_\mathcal{E}, \mathcal{L}_\mathsf{M}, \mu_\mathcal{E})$ be the learning MVDF from entailments framework. For any mvd clause c of a target concept representation $\mathcal{T} \in \mathcal{L}_\mathsf{M}$, one can decide in polynomial time in $|\mathcal{T}|$ whether $c \in \mu_\mathcal{E}(\mathcal{T})$.*

Lemma 32 shows how one can transform the counterexamples. To show Lemma 32, we use the following technical lemma, proved by [16].

Lemma 31 ([16]) *Let $\mathcal{T}$ be a set of mvd clauses formulated in V. If $\mathcal{T} \models V_1 \to V_2 \vee V_3$ then either $\mathcal{T} \models V_1 \to (V_2 \cup \{v\}) \vee V_3$ or $\mathcal{T} \models V_1 \to V_2 \vee (V_3 \cup \{v\})$, where $V_1, V_2, V_3, \{v\} \subseteq V$ and V_2, V_3 are non-empty.*

Lemma 32 *Let $\mathfrak{F}$(MVDF$_\mathcal{Q}$) $= (E_\mathcal{Q}, \mathcal{L}_\mathsf{M}, \mu_\mathcal{Q})$ be the learning MVDF from 2-quasi-Horn framework and $\mathfrak{F}$(MVDF$_\mathcal{E}$) $= (E_\mathcal{E}, \mathcal{L}_\mathsf{M}, \mu_\mathcal{E})$ be the learning MVDF from entailments framework. Assume that the target $\mathcal{T}$ and the hypothesis $\mathcal{H}$ are in variables V and $|\mathcal{H}|$ is polynomial in $|\mathcal{T}|$. If $c \in \mu_\mathcal{Q}(\mathcal{T}) \oplus \mu_\mathcal{Q}(\mathcal{H})$ then one can construct in time polynomial in $|\mathcal{T}|$ an mvd clause c' such that $c' \in \mu_\mathcal{E}(\mathcal{T}) \oplus \mu_\mathcal{E}(\mathcal{H})$.*

Proof. We show how one can transform a 2-quasi-Horn clause $X \to v \vee w$ that is a positive counterexample (in $\mathfrak{F}$(MVDF$_\mathcal{Q}$)) into a positive counterexample (in $\mathfrak{F}$(MVDF$_\mathcal{E}$)). If $\mathcal{T} \models X \to v \vee w$ and $\mathcal{H} \not\models X \to v \vee w$ then we construct an mvd clause as the result of initially setting $W = V \setminus (X \cup \{v, w\})$, $Y = \{v\}$ and $Z = \{w\}$ and then applying the following rule until $X \cup Y \cup Z = V$:

- if $\mathcal{T} \models X \to (Y \cup \{w'\}) \vee Z$, where $w' \in W$, (checked by posing membership queries to $\mathsf{MEM}_{\mathcal{T},\mathfrak{F}(\mathrm{MVDF}_\mathcal{Q})}$, as in Remark 2) then add w' to Y. Otherwise, add w' to Z.

By Lemma 31 either $\mathcal{T} \models X \to (Y \cup \{w'\}) \vee Z$ or $\mathcal{T} \models X \to Y \vee (Z \cup \{w'\})$ must hold. Then, $\mathcal{T} \models X \to Y \vee Z$. As $\{X \to Y \vee Z\} \models X \to v \vee w$, we have that $\mathcal{H} \not\models X \to Y \vee Z$. That is, $X \to Y \vee Z \in \mu_{\mathcal{E}}(\mathcal{T}) \oplus \mu_{\mathcal{E}}(\mathcal{H})$. When $\mathcal{T} \not\models X \to v \vee w$ and $\mathcal{H} \models X \to v \vee w$ the argument is similar but we need to check whether $\mathcal{H} \models X \to (Y \cup \{w'\}) \vee Z$, where $w' \in W$. Since in this case we evaluate the hypothesis, no membership query is necessary to produce a negative counterexample. ❏

Lemmas 30 and 32 show how one can compute, respectively, f_{MEM} and f_{EQ} described in Definition 23, and, so, $\mathfrak{F}$(MVDF$_{\mathcal{Q}}$) polynomial time reduces to $\mathfrak{F}$(MVDF$_{\mathcal{E}}$). The difficulty in showing a reduction in the other direction, from $\mathfrak{F}$(MVDF$_{\mathcal{E}}$) to $\mathfrak{F}$(MVDF$_{\mathcal{Q}}$), is to decide whether the target entails a 2-quasi-Horn clause using polynomially many membership queries with mvd clauses as input.