This is the preprint of the article that appeared in final form in **Theoretical Computer Science** 716 : 4-14 (2018) , which has been published in final form at https://doi.org/10.1016/j.tcs.2017.11.018.

Exact Learning of Multivalued Dependency Formulas

Montserrat Hermo

Languages and Information Systems, University of the Basque Country, Spain

Ana Ozaki

Department of Computer Science, University of Liverpool, United Kingdom

Abstract

The transformation of a relational database schema into fourth normal form, which minimizes data redundancy, relies on the correct identification of multivalued dependencies. In this work, we study the learnability of multivalued dependency formulas (MVDF), which correspond to the logical theory behind multivalued dependencies. As we explain, MVDF lies between propositional Horn and 2-Quasi-Horn. We prove that MVDF is polynomially learnable in Angluin et al.'s exact learning model with membership and equivalence queries, provided that counterexamples and membership queries are formulated as 2-Quasi-Horn clauses. As a consequence, we obtain that the subclass of 2-Quasi-Horn theories which are equivalent to MVDF is polynomially learnable.

Keywords: Exact Learning, Multivalued Dependencies, 2-Quasi-Horn

1. Introduction

Among the models proposed to represent databases, since its presentation by Codd [1], the relational model has been the most successful one. In this model, data is represented by tuples which are grouped into relations. Different types of formalisms based on the concept of data dependencies have been used to design and analyse database schemas. Data dependencies can be classified as functional [1], or multivalued [2, 3], where the latter is a generalization of the first. Functional dependencies correspond to the Horn fragment of propositional logic in the sense that one can map each functional dependency to a Horn clause preserving the logical consequence relation [4, 5]. The same correspondence can be established between multivalued dependencies and multivalued dependency formulas (MVDF) [4, 6]. They have long been studied in the literature and it is well known that the transformation of a relational database schema into the fourth normal form (4NF), which minimizes data redundancy, relies on the identification of multivalued dependencies [3].

In this work, we cast the problem of identifying data dependencies as a learning problem and study the learnability of MVDF, which correspond to the logical theory behind data dependencies. Identification of the Horn fragment from interpretations in Angluin's exact learning model is stated in [7], and later an algorithm that learns Horn from entailments is presented in [8]. Furthermore, a variant that learns sets of functional dependencies appears in [9]. Regarding MVDF, it is known that this class cannot be learned either using equivalence [10] or membership queries alone [11], and that a particular subclass of them is learnable when both types of queries are allowed [12, 13]. However, to the best of our knowledge, there is no positive result for the general class MVDF using membership and equivalence queries. One of main obstacles to find a learning algorithm for MVDF is the fact the MVDF theories are not closed under intersection in contrast to the Horn case [6]. In general, given a multivalued dependency formula, there is not a unique minimal model that satisfies both the formula and a particular set of variables, a property extensively exploited by Horn algorithms.

A major open problem in learning theory (and also within the exact learning model) is whether the class CNF (or the class DNF) can be efficiently learnable. Although it is known that this class cannot be polynomially learned using either membership or equivalence queries alone [14, 10], it is open whether CNF can be learned using both types of queries. Several restrictions have been imposed on both CNF and DNF in order to make them polynomially learnable. For instance, the classes monotone DNF [14], i.e., DNF formulas with no negated variables, k-term DNF or k-clause CNF [15], that is, DNF or CNF formulas with at most k terms or k clauses, and read-twice DNF [16], which are DNF where each variable occurs at most twice, are all polynomially learnable via queries.

One of the most important results concerning a restriction of the class CNF appears in the mentioned article [7], where propositional Horn formulas are learned using both types of queries. In fact, Horn is a special case of a class called k-quasi-Horn, meaning that clauses may contain at most k unnegated literals. However, it is pointed in [7] that, even for $k = 2$, learning the class of k-quasi-Horn formulas is as hard as learning CNF (Corollary 25 of [17]). 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. Since MVDF is a natural restriction of 2-quasi-Horn and a non-trivial generalization of Horn, investigating how far this boundary can be extended constitutes one of our main motivations and guide for this work, which is theoretical in nature.

In this paper, we give a polynomial algorithm that exactly learns MVDF using membership and equivalence queries. Membership queries and counterexamples given by the oracle are formulated as 2-quasi-Horn clauses. As a consequence, an algorithm that efficiently learns the subclass of 2-quasi-Horn formulas which are equivalent to multivalued dependency formulas is obtained. The paper is organized as follows. In Section 2 we introduce some notation and give definitions for MVDF and the class of k-quasi-Horn formulas. Section 3 shows a property that is crucial to learn the class MVDF: (although not unique) the number of minimal models that satisfy a multivalued dependency formula and a set of variables is polynomial in the size of the formula. In Section 4 we present our algorithm that efficiently learns the class MVDF from 2-quasi-Horn clauses. In Section 5 we illustrate the algorithm with an example run. We end in Section 6 with some concluding remarks and open problems.

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 $\mathcal{C}$ 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 $\mathcal{C}$ of concepts over E.

Given a learning framework $(E, \mathcal{L}, \mu)$, for each $l \in \mathcal{L}$, denote by $\mathsf{MEM}_{l,E}$ 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 $\mathsf{MEM}_{l,E}$ with some $e \in E$ as input, for $l \in \mathcal{L}$ and E. Similarly, for every $l \in \mathcal{L}$, we denote by $\mathsf{EQ}_{l,E}$ 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 $\mathsf{EQ}_{l,E}$ 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 *exact 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 $\mathsf{MEM}_{l,E}$ and $\mathsf{EQ}_{l,E}$, respectively. A learning framework $(E, \mathcal{L}, \mu)$ is *polynomial time* exact learnable if it is exact learnable by an 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 Dependencies and k-quasi-Horn Formulas. Let V be a set of boolean variables. The logical constant *true* is represented by $\mathbf{T}$ and the

[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.

logical constant *false* is represented by $\mathbf{F}$. An *mvd clause* is an implication $X \to Y \lor Z$, where X, Y and Z are pairwise disjoint conjunctions of variables from V and $X \cup Y \cup Z = V$. 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. 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 $\varphi = (v_1 \to (v_2 \land v_3) \lor (v_4 \land v_5 \land v_6)) \land ((v_2 \land v_3) \to (v_1 \land v_5 \land v_6) \lor v_4)$ is a formula then we write φ in this shorter way: $\{1 \to 23 \lor 456, 23 \to 156 \lor 4\}$, where conjunctions between variables are omitted and each propositional variable $v_i \in V$ is mapped to $i \in \mathbb{N}$. From the definitions above 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: $\{12356 \to 4, 135 \to 4 \lor 26\}$;

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 $1 \to 23 \lor 456$, by distribution, is equivalent to: $\{1 \to 2 \lor 4, 1 \to 2 \lor 5, 1 \to 2 \lor 6, 1 \to 3 \lor 4, 1 \to 3 \lor 5, 1 \to 3 \lor 6\}$.

Remark. Point 1 above means that w.l.o.g. we can assume that any mvd clause is either $V \to \mathbf{F}$ or $V \setminus \{v\} \to v$ or of the form $X \to Y \lor Z$ with Y and Z non-empty. We call *Horn-like* clauses of the form $V \setminus \{v\} \to v$. Note that $\mathbf{T} \to V \equiv \{\mathbf{T} \to v \mid v \in V\}$ and each $\mathbf{T} \to v$ is equivalent to $\{\mathbf{T} \to V \setminus \{v\} \lor v, V \setminus \{v\} \to v\}$.

Formally, in this paper we study the learning framework $\mathfrak{F}_M = (E_Q, \mathcal{L}_M, \mu_M)$, where E_Q is the set of all 2-quasi-Horn clauses in the propositional variables V under consideration, $\mathcal{L}_M$ is the set of all MVDF that can be expressed in V and, for every $\mathcal{T} \in \mathcal{L}_M$, $\mu_M(\mathcal{T}) = \{e \in E_Q \mid \mathcal{T} \models e\}$. Note that learning MVDF from 2-quasi-Horn examples also corresponds to learning the set of all 2-quasi-Horn formulas that can be constructed by distribution from any mvd formula.

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 $\mathsf{false}(\mathcal{I}) = V \setminus \mathsf{true}(\mathcal{I})$. Let $\mathcal{H}$ and $\mathcal{T}$ be sets of mvd clauses. If $\mathcal{I} \models \mathcal{H}$ and $\mathcal{I} \not\models \mathcal{T}$ then we say that $\mathcal{I}$ is a *negative countermodel* w.r.t. $\mathcal{T}$. We follow the terminology provided in [7] 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) there is v such that $\mathsf{false}(\mathcal{I}) = \{v\}$ and $X \to Y \vee Z$ is the Horn-like clause $V \setminus \{v\} \to v$; or (c) $\mathsf{false}(\mathcal{I}) = \emptyset$ and $X \to Y \vee Z$ is the clause $V \to \mathbf{F}$. Given two interpretations $\mathcal{I}$ and $\mathcal{I}'$, we define $\mathcal{I} \cap \mathcal{I}'$ to be the interpretation such that $\mathsf{true}(\mathcal{I} \cap \mathcal{I}') = \mathsf{true}(\mathcal{I}) \cap \mathsf{true}(\mathcal{I}')$.

3. Computing Minimal Models

In this section, we present Algorithm 1, which computes in polynomial time all minimal models (i.e. models with minimal number of variables assigned to 'true') satisfying both a set $\mathcal{P}$ of mvd clauses and a set of variables X. To ensure the existence of minimal models, we consider $\mathcal{P}$ such that $\mathcal{P}$ does not contain $V \to \mathbf{F}$. Algorithm 1 receives $\mathcal{P}$ and X as input and constructs a *semantic tree*, in the sense that each child node satisfies one of the two consequents of an mvd clause. In each iteration of the main loop we 'apply' an mvd clause, meaning that, given a tree leaf node, we pick a (not used) mvd clause $X' \to Y' \vee Z' \in \mathcal{P}$ and construct two child nodes, one of them containing variables in Y' and the other variables in Z'. We exhaustively apply mvd clauses in $\mathcal{P}$ so that in the end each leaf node contains a set of variables that need to be true in order to satisfy both X and $\mathcal{P}$. Note that Horn-like clauses are treated in Line 16.

The following information is stored for each node i: a set M_i of mvd clauses in $\mathcal{P}$ that have not yet been applied in the i-node path; and a set S_i of variables implied by X and by mvd clauses that have already been applied (i.e. clauses in $\mathcal{P} \setminus M_i$). The following example illustrates how the algorithm works.

Algorithm 1 Semantic Tree

```
Let 𝒮 = ∅ be a set of interpretations
Let 𝒫 be a set of mvd clauses without V → F and X a set of variables
function SemanticTree(𝒫, X)
    Create a node i = 0 with S_0 = X and M_0 = 𝒫
    repeat
        if there is a leaf i and an mvd clause X' → Y' ∨ Z' ∈ M_i with Y' ≠ ∅
                Z' ≠ ∅ and X' ⊆ S_i then
            if there is v ∈ Y' ∪ Z' such that v ∉ S_i then
                Create a new node 2i + 1 as a child of i
                S_{2i+1} = S_i ∪ Y', M_{2i+1} = M_i \ {X' → Y' ∨ Z'}
                Create a new node 2i + 2 as a child of i
                S_{2i+2} = S_i ∪ Z', M_{2i+2} = M_i \ {X' → Y' ∨ Z'}
            end if
        end if
    until no more nodes can be created
    for every node j that is a leaf do
        Create ℐ with true(ℐ) = S_j ∪ {v | S_j = V \ {v}, V \ {v} → v ∈ 𝒫}
        𝒮 := 𝒮 ∪ {ℐ}
    end for
    return (𝒮)
end function
```

Example Let $X = \{1,2,3,4\}$ and $\mathcal{P} = \{c_1 = 13 \to 257 \vee 468, c_2 = 12 \to 34 \vee 5678, c_3 = 145 \to 26 \vee 378, c_4 = 1234567 \to 8\}$ be a set of mvd clauses. In Line 6, the choice of an mvd clause made by Algorithm 1 is non-deterministic and in this example we choose clauses in the same order they appear above. The root of the semantic tree of $\mathcal{P}$ and X has $S_0 = \{1,2,3,4\}$ and $M_0 = \{c_1, c_2, c_3, c_4\}$. Choosing the first clause c_1 we have $S_1 = \{1,2,3,4,5,7\}$, $S_2 = \{1,2,3,4,6,8\}$ and $M_1 = M_2 = \{c_2, c_3, c_4\}$. Now we choose the second clause c_2 to obtain $S_3 = S_1$ $S_4 = \{1,2,3,4,5,6,7,8\}$ $S_5 = S_2$ $S_6 = S_4$ and $M_3 = M_4 = M_5 = M_6 = \{c_3, c_4\}$.

Finally, we choose third clause. Figure 1 illustrates the Semantic Tree with variables which are 'new' in the path, that is, if node a is predecessor of node i in the tree then S_a does not have these variables. In Line 16, Algorithm 1 checks that the antecedent of c_4 is satisfied in node 7 and adds variable 8 to its corresponding interpretation. Algorithm 1, returns $\mathcal{S} = \{\mathcal{I}_1, \mathcal{I}_2, \mathcal{I}_3\}$, with $\mathsf{true}(\mathcal{I}_1) = \{1,2,3,4,5,6,7,8\}$, $\mathsf{true}(\mathcal{I}_2) = \{1,2,3,4,5,7,8\}$ and $\mathsf{true}(\mathcal{I}_3) = \{1,2,3,4,6,8\}$.

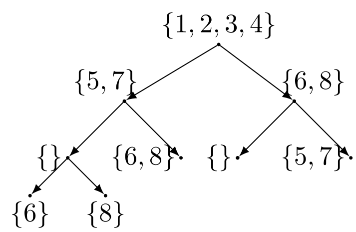

Figure 1: Semantic Tree

The following Theorem shows that Algorithm 1 runs in polynomial time and that the returned set $\mathcal{S}$ includes all minimal models satisfying $\mathcal{P}$ and X.

Theorem 1 *Let $\mathcal{P}$ be a set of mvd clauses and X a set of variables. One can construct in polynomial time w.r.t. $|\mathcal{P}|$ a set of interpretations $\mathcal{S}$ that verifies the following properties:*

1. *if $\mathcal{I} \in \mathcal{S}$ then $\mathcal{I} \models \mathcal{P}$;*
2. *if $\mathcal{I}' \models \mathcal{P}$ and $X \subseteq \mathsf{true}(\mathcal{I}')$ then there is $\mathcal{I} \in \mathcal{S}$ such that $\mathsf{true}(\mathcal{I}) \subseteq \mathsf{true}(\mathcal{I}')$.*

Proof. Let $\mathcal{S}$ be the return value of Algorithm 1 with $\mathcal{P}$ and X as input. Point (1) is a corollary of a more general one: for all nodes i in a semantic tree, the interpretation $\mathcal{I}'$ defined as $\mathsf{true}(\mathcal{I}') = S_i$ is a model of the set of mvd clauses

$X' \to Y' \vee Z' \in \mathcal{P} \setminus M_i$ with Y' and Z' non-empty. The proof of this fact is by induction in the number of levels of the semantic tree. (Algorithm 1 treats Horn-like clauses in Line 16.) Point (2) is a corollary of a more general one: if $\mathcal{I}' \models \mathcal{P}$ and $X \subseteq \mathsf{true}(\mathcal{I}')$, then there exists a path from the root to a node k that is leaf, in such a way that $S_k \subseteq \mathsf{true}(\mathcal{I}')$. The proof of this fact is again by induction in the number of levels of the semantic tree.

Now, it remains to show that the construction of $\mathcal{S}$ is in polynomial time. Let $n = |V|$ and $m = |\mathcal{P}|$. The fact that the number of nodes of any semantic tree for $\mathcal{P}$ and X is bounded by $2 \times n \times m$ follows from the next 3 claims. This bound implies that the running time of the construction is polynomial in $|V|$ and $|\mathcal{P}|$. To simplify the presentation, we use auxiliary sets N_i which contain variables that are 'new' in the set S_i of a node i in the semantic tree, that is, if node a is predecessor of node i in the tree then S_a does not have these variables.

Claim 1 For any level j whose nodes are $j_1, j_2, \ldots j_k$, the sets $N_{j_1}, N_{j_2}, \ldots, N_{j_k}$ are pairwise disjoint.

Let j_1 and j_2 be arbitrary distinct nodes at level j and let a be the lowest common ancestor of nodes j_1 and j_2. For all $v \in N_{j_1} \cup N_{j_2}$, we have that $v \notin S_a$. Let b_1 and b_2 be the two children of the ancestor a. In the construction of the semantic tree, Algorithm 1 introduces v in exactly one of S_{b_1} or S_{b_2}. Therefore, it is not possible to find $v \in N_{j_1} \cap N_{j_2}$.

Claim 2 For any level j whose nodes are $j_1, j_2, \ldots j_k$ there are at most $k/2$ nodes such that the corresponding sets $N_{j_1}, N_{j_2}, \ldots, N_{j_k}$ are empty.

In the execution of Algorithm 1, whenever two children of a node are created, at least one new variable is introduced in at least one of the siblings.

Thus, the number of nodes in each level is bounded by $2 \times n$.

Claim 3 The depth of a semantic tree for $\mathcal{P}$ is bounded by m.

This is because any mvd clause is used at most once along a branch. Note that this claim also ensures termination. ❏

It is worth saying that Algorithm 1 allows us to decide whether a set of mvd clauses $\mathcal{P}$ is satisfiable. Note that if $V \to \mathbf{F} \notin \mathcal{P}$ then $\mathcal{P}$ is trivially satisfiable. Otherwise, we only need to set the input X as empty and check whether $\mathcal{S}$ (the return of Algorithm 1) contains only $\mathcal{I}$ such that $\mathsf{true}(\mathcal{I}) = V$. If so then $\mathcal{P} \cup \{V \to \mathbf{F}\}$ is unsatisfiable.

4. Learning MVDF from 2-quasi-Horn

In this section we present an algorithm that learns the class MVDF from 2-quasi-Horn. More precisely, we show that the learning framework $\mathfrak{F}_M = (E_Q, \mathcal{L}_M, \mu_M)$ (defined in the Preliminaries) is exact learnable in polynomial time.

Algorithm 2 maintains a sequence $\mathfrak{L}$ of distinct interpretations used to construct the learner's hypothesis $\mathcal{H}$. We use $\mathcal{I}_i \in \mathfrak{L}$ to denote an interpretation at the i-th position in the sequence $\mathfrak{L}$. In each iteration of the main loop, if $\mathcal{H}$ is not equivalent to the target $\mathcal{T}$ then the oracle $\mathsf{EQ}_{\mathcal{T},E_Q}$ provides the learner with a 2-quasi-Horn clause c that is a positive counterexample. That is, $\mathcal{T} \models c$ and $\mathcal{H} \not\models c$. The assumption that the counterexample is positive is justified by the construction of $\mathcal{H}$, which ensures at all times that $\mathcal{T} \models \mathcal{H}$. Each *positive* counterexample is used to construct a *negative* countermodel that either refines an element of $\mathfrak{L}$, or is added to the end of $\mathfrak{L}$. In order to learn all of the clauses in $\mathcal{T}$, we would like the clauses induced by the elements in $\mathfrak{L}$ to approximate distinct clauses in $\mathcal{T}$. This will happen if at most polynomially many elements in $\mathfrak{L}$ violate the same clause in $\mathcal{T}$. As explained in [7], overzealous refinement may result in several elements in $\mathfrak{L}$ violating the same clause of $\mathcal{T}$. This is avoided in Algorithm 2 by (1) refining at most one (the first) element of $\mathfrak{L}$ per iteration and (2) previously refining the constructed countermodel with the elements of $\mathfrak{L}$.

The following notion essentially describes under which conditions we say that it is 'good' to refine an interpretation (which can be either a countermodel or an element of $\mathfrak{L}$). There are two cases this can happen in our algorithm: (1) an element in $\mathfrak{L}$ is refined with a countermodel (Line 10 of Algorithm 2) or (2) the countermodel is refined with some element in $\mathfrak{L}$ (Line 5 of Algorithm 3).

Definition 1 *We say that 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)* $\mathcal{I} \cap \mathcal{I}' \not\models \mathcal{T}$.

Algorithms for Horn formulas in [7, 8] use a notion of 'goodCandidate' that is more relaxed than ours. They only need conditions (i) and (iii). The reason is that (ii) always holds because the intersection of two models of a set of Horn clauses $\mathcal{H}$ is also a model of $\mathcal{H}$. The lack of this property in the case of MVDF has two consequences. The first one is that there is not a unique minimal model that satisfies both an mvd formula and a particular set of variables. We solved this problem in the previous section, by constructing a semantic tree. The second consequence is that in the Horn algorithm [7] only a single interpretation of the sequence that the algorithm maintains can violate a Horn clause from the target. However, in our algorithm any mvd clause of the target $\mathcal{T}$ can be violated by polynomially many interpretations of the sequence $\mathfrak{L}$. Function 'RefineCounterModel' (Algorithm 3) is crucial to ensure that only polynomially many interpretations in $\mathfrak{L}$ violate the same clause in $\mathcal{T}$. We later show that in any run only a polynomial amount of interpretations can be removed from $\mathfrak{L}$.

Remark. In the rest of this paper we consider interpretations $\mathcal{I}$ such that $|\mathsf{false}(\mathcal{I})| \geq 1$. This is justified by the fact that in Line 1 of Algorithm 2 we check whether $\mathcal{T} \models V \to \mathbf{F}$ and if so we add it to $\mathcal{H}_0$. Then, any negative countermodel $\mathcal{I}$ computed in Line 6 is such that $|\mathsf{false}(\mathcal{I})| \geq 1$.

Lemmas 1, 2 and 4 show how Algorithm 2 can be implemented. In particular, Lemma 1 shows how the learner can decide Point (iii) of Definition 1 with polynomially many 2-quasi-Horn entailment queries. It also shows how Line 6 of Algorithm 2 can be implemented.

Algorithm 2 Learning algorithm for MVDF from 2-quasi-Horn clauses

Let $\mathcal{H}_0 = \{V \rightarrow \mathbf{F} \mid \mathcal{T} \models V \rightarrow \mathbf{F}\}$, $\mathfrak{L} = \emptyset$, $\mathcal{H} = \mathcal{H}_0$
Let BuildClauses($\mathcal{I}$) be the function that given an interpretation $\mathcal{I}$ with $|\mathsf{false}(\mathcal{I})| \geq 1$ returns $\{\mathsf{true}(\mathcal{I}) \rightarrow v \vee w \mid v \neq w \in \mathsf{false}(\mathcal{I}),\ \mathcal{T} \models \mathsf{true}(\mathcal{I}) \rightarrow v \vee w\} \cup \{\mathsf{true}(\mathcal{I}) \rightarrow v \mid \mathsf{true}(\mathcal{I}) = V \setminus \{v\}, \mathcal{T} \models \mathsf{true}(\mathcal{I}) \rightarrow v\}$
while $\mathcal{H} \not\equiv \mathcal{T}$ **do**
    Let $X \rightarrow v \vee w$ (or $X \rightarrow v$) be a 2-quasi-Horn positive counterexample
    Let $\mathcal{S}$ be the return value of SemanticTree($\mathcal{H}$, X)
    Find $\mathcal{I} \in \mathcal{S}$ such that $\mathcal{I} \not\models \mathcal{T}$ (we know that for all $\mathcal{I} \in \mathcal{S}$, $\mathcal{I} \models \mathcal{H}$)
    $\mathcal{J}$:= RefineCounterModel($\mathcal{I}$)
    if there is $\mathcal{I}_k \in \mathfrak{L}$ such that $\mathsf{goodCandidate}(\mathcal{I}_k, \mathcal{J})$ **then**
        Let $\mathcal{I}_i$ be the first in $\mathfrak{L}$ such that $\mathsf{goodCandidate}(\mathcal{I}_i, \mathcal{J})$
        Replace $\mathcal{I}_i$ by $\mathcal{J}$
        Remove all $\mathcal{I}_j \in \mathfrak{L} \setminus \{\mathcal{J}\}$ such that $\mathcal{I}_j \not\models$ BuildClauses($\mathcal{J}$)
    else
        Append $\mathcal{J}$ to $\mathfrak{L}$
    end if
    Construct $\mathcal{H} = \mathcal{H}_0 \cup$TransformMVDF($\bigcup_{\mathcal{I} \in \mathfrak{L}}$BuildClauses($\mathcal{I}$))
end while

Lemma 1 *Let $\mathcal{I}$ be an interpretation and $\mathcal{T}$ the target (set of mvd clauses). One can decide in polynomial time whether $\mathcal{I}$ satisfies $\mathcal{T}$ using polynomially many 2-quasi-Horn entailment queries.*

Proof. Let $C = \{\mathsf{true}(\mathcal{I}) \rightarrow v \vee w \mid v, w \in \mathsf{false}(\mathcal{I}),\ \mathcal{T} \models \mathsf{true}(\mathcal{I}) \rightarrow v \vee w\} \cup \{\mathsf{true}(\mathcal{I}) \rightarrow v \mid \mathsf{true}(\mathcal{I}) = V \setminus \{v\}, \mathcal{T} \models \mathsf{true}(\mathcal{I}) \rightarrow v\}$. We show that $\mathcal{I}$ does not satisfy $\mathcal{T}$ if, and only if, there is $c \in C$ such that $\mathcal{T} \models c$. ($\Rightarrow$) If $\mathcal{I}$ does not satisfy $\mathcal{T}$ then there is $X \rightarrow Y \vee Z \in \mathcal{T}$ that is violated by $\mathcal{I}$. That is, $X \subseteq \mathsf{true}(\mathcal{I})$ 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) there is v such that $\mathsf{false}(\mathcal{I}) = \{v\}$ and $X \rightarrow Y \vee Z$ is the Horn-like clause $V \setminus \{v\} \rightarrow v$. In case (a) we have that $\mathcal{T} \models \mathsf{true}(\mathcal{I}) \rightarrow v \vee w$.

Algorithm 3 Function to refine the countermodel

function REFINECOUNTERMODEL($\mathcal{I}$)
    Let $\mathcal{J} := \mathcal{I}$
    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 $\mathsf{goodCandidate}(\mathcal{I}, \mathcal{I}_i)$
        $\mathcal{J}$:=REFINECOUNTERMODEL($\mathcal{I} \cap \mathcal{I}_i$)
    end if
    return ($\mathcal{J}$)
end function

In case (b) we have $\mathcal{T} \models \mathsf{true}(\mathcal{I}) \rightarrow v$ (meaning that $\mathcal{T} \models V \setminus \{v\} \rightarrow v$). ($\Leftarrow$) Follows from the fact that for all $c \in C$, $\mathcal{I} \not\models c$. ❏

In Line 5 Algorithm 2 calls a function that builds a semantic tree (Algorithm 1) for $\mathcal{H}$ and the antecedent of the counterexample given in Line 4. Using this tree, by Lemma 2, one can create in polynomial time a set of interpretations $\mathcal{S}$ such that (i) for all $\mathcal{I} \in \mathcal{S}$, $\mathcal{I} \models \mathcal{H}$ and (ii) there is an interpretation $\mathcal{I} \in \mathcal{S}$ such that $\mathcal{I} \not\models \mathcal{T}$. That is, there is $\mathcal{I} \in \mathcal{S}$ such that $\mathcal{I}$ is a negative countermodel. By Theorem 1, $\mathcal{S}$ is computed in polynomial time w.r.t. $|\mathcal{H}|$ (we show later that $|\mathcal{H}|$ is polynomial in $|\mathcal{T}|$).

Lemma 2 *Let X be the set of variables in the antecedent of a positive counterexample c received in Line 4 of Algorithm 2. Let $\mathcal{S}$ be the return of Algorithm 1 with $\mathcal{H}$ and X as the input. All interpretations in $\mathcal{S}$ satisfy $\mathcal{H}$ and at least one interpretation of $\mathcal{S}$ does not satisfy $\mathcal{T}$.*

Proof. $\mathcal{S}$ verifies the following properties: (1) if $\mathcal{I} \in \mathcal{S}$ then $\mathcal{I} \models \mathcal{H}$; (2) if $\mathcal{I}' \models \mathcal{H}$ and $X \subseteq \mathsf{true}(\mathcal{I}')$ then there is $\mathcal{I} \in \mathcal{S}$ such that $\mathsf{true}(\mathcal{I}) \subseteq \mathsf{true}(\mathcal{I}')$; and (3) there is an interpretation $\mathcal{I} \in \mathcal{S}$ such that $\mathcal{I} \not\models \mathcal{T}$. By Theorem 1 we have Points (1) and (2). For Point (3), we show that there is an interpretation $\mathcal{I} \in \mathcal{S}$ such that $\mathcal{I} \not\models \mathcal{T}$. As $\mathcal{H} \not\models c$, there is an interpretation $\mathcal{I}'$ such that $\mathcal{I}' \models \mathcal{H}$ and $\mathcal{I}' \not\models c$. Thus, $X \subseteq \mathsf{true}(\mathcal{I}')$ and by Points (1) and (2), there is $\mathcal{I} \models \mathcal{H}$ such that

Algorithm 4 Transform a 2-quasi-Horn clause into an mvd clause

1: **function** TRANSFORMMVDF($\mathcal{H}'$)
2: $\quad \mathcal{H} := \{c \in \mathcal{H}' \mid c$ is of the form $V \setminus \{v\} \to v\}$
3: $\quad$ **for** every $X \to v \vee w \in \mathcal{H}'$ **do**
4: $\quad\quad$ Let $W = V \setminus (X \cup \{v, w\})$, $Y = \{v\}$ and $Z = \{w\}$
5: $\quad\quad$ **for** each $w' \in W$ **do**
6: $\quad\quad\quad$ **if** $\mathcal{T} \models X \to Y\{w'\} \vee Z$ **then**
7: $\quad\quad\quad\quad$ add w' to Y
8: $\quad\quad\quad$ **else**
9: $\quad\quad\quad\quad$ add w' to Z
10: $\quad\quad\quad$ **end if**
11: $\quad\quad$ **end for**
12: $\quad\quad$ add $X \to Y \vee Z$ to $\mathcal{H}$
13: $\quad$ **end for**
14: $\quad$ **return** $(\mathcal{H})$
15: **end function**

$\mathsf{true}(\mathcal{I}) \subseteq \mathsf{true}(\mathcal{I}')$. Since for all $\mathcal{I} \in \mathcal{S}$, we have that $X \subseteq \mathsf{true}(\mathcal{I})$, the latter fact ensures that $\mathcal{I} \not\models c$ and therefore $\mathcal{I} \not\models \mathcal{T}$. ❑

Given a set of 2-quasi-Horn clauses, Algorithm 4 transforms each 2-quasi-Horn clause c into an mvd clause c' such that $\{c'\} \models c$ with polynomially many 2-quasi-Horn entailment queries. This property is exploited by the learner to generate mvd clauses for the hypothesis in Line 15 of Algorithm 2. Lines 5-11 of Algorithm 4 rely on Lemma 4. Lemma 4 requires the following technical lemma.

Lemma 3 *Let $\mathcal{T}$ be a set of mvd clauses. If $\mathcal{I}$ and $\mathcal{I}'$ are models such that $\mathcal{I} \models \mathcal{T}$ and $\mathcal{I}' \models \mathcal{T}$, but $\mathcal{I} \cap \mathcal{I}' \not\models \mathcal{T}$, then true$(\mathcal{I}) \cup$ true$(\mathcal{I}') = V$.*

Proof. If $\mathcal{I} \models \mathcal{T}$, $\mathcal{I}' \models \mathcal{T}$ and $\mathcal{I} \cap \mathcal{I}' \not\models \mathcal{T}$ then there is $X \to Y \vee Z \in \mathcal{T}$ such that $X \subseteq \mathsf{true}(\mathcal{I} \cap \mathcal{I}')$ and either $Y \subseteq \mathsf{true}(\mathcal{I})$ and $Z \subseteq \mathsf{true}(\mathcal{I}')$; or; $Z \subseteq \mathsf{true}(\mathcal{I})$ and $Y \subseteq \mathsf{true}(\mathcal{I}')$. Assume $Y \subseteq \mathsf{true}(\mathcal{I})$ and $Z \subseteq \mathsf{true}(\mathcal{I}')$ (the other case is symmetric). Given $v \in V$, we know that it has to be either in X, Y or Z. If $v \in X$ then $v \in \mathsf{true}(\mathcal{I} \cap \mathcal{I}')$. If $v \in Y$ then $v \in \mathsf{true}(\mathcal{I})$. Also, if $v \in Z$ then $v \in \mathsf{true}(\mathcal{I}')$. In all cases it holds that $v \in \mathsf{true}(\mathcal{I}) \cup \mathsf{true}(\mathcal{I}')$. Then, $\mathsf{true}(\mathcal{I}) \cup \mathsf{true}(\mathcal{I}') = V$. ❏

Lemma 4 *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 $\mathcal{T} \models V_1 \to V_2\{v\} \vee V_3$ or $\mathcal{T} \models V_1 \to V_2 \vee V_3\{v\}$, where $V_1, V_2, V_3, \{v\} \subseteq V$ and V_2, V_3 are non-empty*[2]*.*

Proof. We can assume that $v \notin V_1$, otherwise the lemma trivially holds. Now, assuming that $\mathcal{T} \models V_1 \to V_2 \vee V_3$ and $\mathcal{T} \not\models V_1 \to V_2\{v\} \vee V_3$, we prove that $\mathcal{T} \models V_1 \to V_2 \vee V_3\{v\}$ (the other case is symmetric). The proof is by showing that any model $\mathcal{I}$ of $\mathcal{T}$ satisfies $V_1 \to V_2 \vee V_3\{v\}$. Suppose that $\mathcal{I} \models \mathcal{T}$ and $V_1 \subseteq \mathsf{true}(\mathcal{I})$, otherwise we are done. Then either $V_3 \subseteq \mathsf{true}(\mathcal{I})$ or $V_2 \subseteq \mathsf{true}(\mathcal{I})$. The latter means that $\mathcal{I}$ already satisfies $V_1 \to V_2 \vee V_3\{v\}$. So it remains to see the case when $V_2 \not\subseteq \mathsf{true}(\mathcal{I})$ and $V_3 \subseteq \mathsf{true}(\mathcal{I})$. If $\mathcal{T} \not\models V_1 \to V_2\{v\} \vee V_3$ there is $\mathcal{I}' \models \mathcal{T}$ such that $V_1 \subseteq \mathsf{true}(\mathcal{I}')$ with $V_2\{v\} \not\subseteq \mathsf{true}(\mathcal{I}')$ and $V_3 \not\subseteq \mathsf{true}(\mathcal{I}')$. As $\mathcal{I} \cap \mathcal{I}' \not\models \mathcal{T}$, by Lemma 3, $\mathsf{true}(\mathcal{I}) \cup \mathsf{true}(\mathcal{I}') = V$. Since $\mathcal{I}' \models \mathcal{T}$ and $\mathcal{T} \models V_1 \to V_2 \vee V_3$, the only option is $v \notin \mathsf{true}(\mathcal{I}')$, so $v \in \mathsf{true}(\mathcal{I})$. As $\mathcal{I} \models \mathcal{T}$ and $\mathcal{T} \models V_1 \to V_2 \vee V_3$, if $v \in \mathsf{true}(\mathcal{I})$ then $\mathcal{I} \models V_1 \to V_2 \vee V_3\{v\}$. So any model of $\mathcal{T}$ must satisfy $V_1 \to V_2 \vee V_3\{v\}$. Then, $\mathcal{T} \models V_1 \to V_2 \vee V_3\{v\}$. ❏

If Algorithm 2 terminates, then it obviously has found a hypothesis $\mathcal{H}$ that is logically equivalent to $\mathcal{T}$. It thus remains to show that the algorithm terminates in polynomial time. We can see the hypothesis $\mathcal{H}$ as a sequence of sets of entailments, where each $\mathcal{H}_i$ corresponds to the transformation of the set "BuildClauses" with $\mathcal{I}_i$ in $\mathfrak{L}$ as input into mvd clauses (Line 15 of Algorithm 2). By Line 2 of Algorithm 2 the number of entailments created by "BuildClauses" is bounded by $|V|^2 + 1$. Lemmas 5 to 10 show that at all times the number of interpretations in $\mathfrak{L}$ that violate a clause in $\mathcal{T}$ is bounded by $|V|$.

[2] $V_i\{v\}$ is the conjunction of variables in V_i and v, where $i \in \{2, 3\}$

From now on, we denote the sequence produced by Algorithm 2 as $\mathfrak{L}$. We always assume that $i \neq j$ when $\mathcal{I}_i$ and $\mathcal{I}_j$ are interpretations in $\mathfrak{L}$.

Lemma 5 *Assume that an interpretation $\mathcal{I}$ violates $c \in \mathcal{T}$. For all $\mathcal{I}_i \in \mathfrak{L}$ such that $\mathcal{I}_i$ covers c, $\mathsf{true}(\mathcal{I}_i) \subseteq \mathsf{true}(\mathcal{I})$ if, and only if, $\mathcal{I} \not\models$ BuildClauses($\mathcal{I}_i$).*

Proof. Let c be the mvd clause $X \to Y \vee Z$. 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: (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) there is v such that $\mathsf{false}(\mathcal{I}) = \{v\}$ and $X \to Y \vee Z$ is the Horn-like clause $V \setminus \{v\} \to v$. In case (a), as $\mathsf{true}(\mathcal{I}_i) \subseteq \mathsf{true}(\mathcal{I})$, we have that $v \in Y \setminus \mathsf{true}(\mathcal{I}_i)$ and $w \in Z \setminus \mathsf{true}(\mathcal{I}_i)$. Since $\mathcal{I}_i$ covers $X \to Y \vee Z$, $X \subseteq \mathsf{true}(\mathcal{I}_i)$.Then $\mathcal{T} \models \mathsf{true}(\mathcal{I}_i) \to v \vee w$. By definition of $\mathcal{H}_i$, there is $X_i \to Y_i \vee Z_i \in \mathcal{H}_i$ such that $v \in Y_i$ and $w \in Z_i$. But this means that $\mathcal{I} \not\models$ BuildClauses($\mathcal{I}_i$). Case (b) is similar, we have that $\{v\} = \mathsf{false}(\mathcal{I}_i)$ and $V \setminus \{v\} = \mathsf{true}(\mathcal{I}_i)$. Then $\mathcal{T} \models \mathsf{true}(\mathcal{I}_i) \to v$ and we also have $\mathcal{I} \not\models$ BuildClauses($\mathcal{I}_i$). ❑

Lemma 6 *At the end of each iteration, for all $\mathcal{I}_i \in \mathfrak{L}$, $\mathcal{I}_i \models \mathcal{H} \setminus \mathcal{H}_i$.*

Proof. By Lemma 2 and Algorithm 3, the interpretation $\mathcal{J}$ computed in Line 7 of Algorithm 2 is a negative countermodel. So if Algorithm 2 executes Line 13 then it holds that $\mathcal{J} \models \mathcal{H}$. If there exists $\mathcal{I}_j \in \mathfrak{L}$ such that $\mathcal{I}_j \not\models$ BuildClauses($\mathcal{J}$), 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 the algorithm did not replace some interpretation in $\mathfrak{L}$. Otherwise, Algorithm 2 executes Line 10 and, then, an interpretation $\mathcal{I}_i \in \mathfrak{L}$ is replaced with $\mathcal{J}$, 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($\mathcal{J}$), but this is always true because of Line 11. ❑

Lemma 7 *If Algorithm 2 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 2 replaced $\mathcal{I}_i \in \mathfrak{L}$ 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 3 of Algorithm 3, 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 8 *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 5, $\mathsf{true}(\mathcal{I}_i) \subseteq \mathsf{true}(\mathcal{I}_j)$ if, and only if, $\mathcal{I}_i \not\models$BuildClauses$(\mathcal{I}_j)$. If $\mathcal{I}_i$ covers $c \in \mathcal{T}$ and $\mathcal{I}_j$ violates $c \in \mathcal{T}$ then it follows from Lemma 6 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 6, it holds that $\mathcal{I}_i \models \mathcal{H} \setminus \{\mathcal{H}_i, \mathcal{H}_j\}$ and $\mathcal{I}_j \models \mathcal{H} \setminus \{\mathcal{H}_i, \mathcal{H}_j\}$. Now, by Lemma 3, $\mathsf{false}(\mathcal{I}_j) \cap \mathsf{false}(\mathcal{J}) \neq \emptyset$ implies that $\mathcal{I}_i \cap \mathcal{I}_j \models \mathcal{H} \setminus \{\mathcal{H}_i, \mathcal{H}_j\}$. 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 9 *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 7 of Algorithm 2. Consider the possibilities.

- If Algorithm 2 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 8 would be satisfied and, instead of appending $\mathcal{J}$, Algorithm 2 would replace some interpretation $\mathcal{I}_k \in \mathfrak{L}$.

- Now assume that Algorithm 2 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 3 of Algorithm 3, 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 9 of Algorithm 2, 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 2 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 6), we have that $\mathcal{I}_j \models \mathcal{H}'_k$. By the same argument $\mathcal{I}_i \models \mathcal{H}'_k$. Hence, by Lemma 3, $\mathsf{false}(\mathcal{I}_i) \cap \mathsf{false}(\mathcal{I}_j) = \emptyset$.

❏

Lemma 10 *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. We assume w.l.o.g. that $i < j$. On the one hand, by Lemma 8 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 hand, by

Lemma 9 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 $\mathsf{false}(\mathcal{I}_i) \cap \mathsf{false}(\mathcal{I}_j) = \emptyset$. ❏

By Lemma 10 if any two interpretations $\mathcal{I}_i, \mathcal{I}_j \in \mathfrak{L}$ violate the same clause in $\mathcal{T}$ then their sets of false variables are disjoint. As for each interpretation $|\mathsf{false}(\mathcal{I})| \geq 1$, the number of mutually disjoint interpretations violating any mvd clause in $\mathcal{T}$ is bounded by $|V|$. Since every $\mathcal{I}_i \in \mathfrak{L}$ is such that $\mathcal{I}_i \not\models \mathcal{T}$, we have that every $\mathcal{I}_i \in \mathfrak{L}$ violates at least one $c \in \mathcal{T}$. This bounds the number of elements in $\mathfrak{L}$ to the number of mvd clauses in $\mathcal{T}$.

Corollary 1 *At the end of each iteration every $c \in \mathcal{T}$ is violated by at most $|V|$ interpretations $\mathcal{I}_i \in \mathfrak{L}$.*

Then, at all times the number of elements in $\mathfrak{L}$ is bounded by $|\mathcal{T}| \cdot |V|$. As in each replacement the number of variables in the antecedent is strictly smaller, we have that the number of replacements that can be done for each $\mathcal{I}_i \in \mathfrak{L}$ is bounded by the number of variables, $|V|$. To ensure the progress of the algorithm, we also need to show that the number of iterations is polynomial in the size of $\mathcal{T}$.

The rest of this section is devoted to show an upper bound polynomial in $|\mathcal{T}|$ on the total number of iterations of Algorithm 2. Before showing our upper bound in Lemma 12, we need the following two lemmas. Essentially, Lemma 9 states the main property obtained by our refinement conditions (Definition 1). Lemma 11 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.

Lemma 11 *In Line 11 of Algorithm 2, 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 10) 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 10) then* $\mathsf{false}(\mathcal{I}_j) \cap \mathsf{false}(\mathcal{I}_k) = \emptyset$.

Proof. First we 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 obtain 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 9 of Algorithm 2, 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 9.

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): By Lemma 7, $\mathsf{true}(\mathcal{J}) \subseteq \mathsf{true}(\mathcal{I}_i)$. If $\mathcal{I}_j$ is removed, then $\mathcal{I}_j \not\models$ BuildClauses($\mathcal{J}$), and there is $c \in$ BuildClauses($\mathcal{J}$) such that $\mathcal{I}_j$ and $\mathcal{J}$ violate c. Then, $\mathcal{I}_i$ covers c. Now assume to a contradiction that $\mathsf{true}(\mathcal{I}_i) \subseteq \mathsf{true}(\mathcal{I}_j)$. Since $\mathcal{I}_i$ covers c and $\mathcal{I}_j$ violates c, we have that $\mathcal{I}_i$ violates c. As $\mathcal{I}_j$ violates c, we obtain that $\mathcal{I}_j \not\models$ BuildClauses($\mathcal{I}_i$), which is a contradiction with Lemma 6. One can give a similar argument to show that $\mathsf{true}(\mathcal{I}_j) \not\subseteq \mathsf{true}(\mathcal{I}_k)$.

- For condition (ii): As $\mathcal{I}_j \models \mathcal{H} \setminus \mathcal{H}_j$ (by Lemma 6) 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 3, $\mathcal{I}_i \cap \mathcal{I}_j \models \mathcal{H} \setminus (\mathcal{H}_i \cup \mathcal{H}_j)$. In fact, by Claim 2, we actually have $\mathcal{I}_i \cap \mathcal{I}_j \models \mathcal{H}$. With a similar argument one can show that $\mathcal{I}_j \cap \mathcal{I}_k \models \mathcal{H}$.

- For condition (iii): As $\mathcal{I}_j \not\models$BuildClauses($\mathcal{J}$) there is $c \in$ BuildClauses($\mathcal{J}$) such that $\mathcal{I}_j$ and $(\mathcal{I}_i \cap \mathcal{J}) = \mathcal{J}$ (by Lemma 7) violate c. Then $\mathcal{I}_i$ covers c, meaning that $\mathcal{I}_i \cap \mathcal{I}_j \not\models c$. By definition of BuildClauses($\mathcal{J}$), $\mathcal{T} \models c$ and, so, $\mathcal{I}_i \cap \mathcal{I}_j \not\models \mathcal{T}$. One can give a similar argument to show that $\mathcal{I}_k \cap \mathcal{I}_j \not\models \mathcal{T}$.

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 9. Then, $\mathsf{false}(\mathcal{I}_i) \cap \mathsf{false}(\mathcal{I}_j) = \emptyset$ and $\mathsf{false}(\mathcal{I}_j) \cap \mathsf{false}(\mathcal{I}_k) = \emptyset$. ❑

<table>
<tr><th>if</th><th>then</th><th>Lemma</th></tr>
<tr><td>$\exists c : (\mathcal{I} \models c) \wedge (\mathcal{I}' \models c) \wedge (\mathcal{I} \cap \mathcal{I}' \not\models c)$</td><td>$\mathsf{false}(\mathcal{I}) \cap \mathsf{false}(\mathcal{I}') = \emptyset$</td><td>3</td></tr>
<tr><td>$\mathcal{I}_i$ is replaced by $\mathcal{J}$</td><td>$\mathsf{false}(\mathcal{I}_i) \subseteq \mathsf{false}(\mathcal{J})$</td><td>7</td></tr>
<tr><td>$\exists c : (\mathcal{I}_i \not\models c) \wedge (\mathcal{I}_j \not\models c)$</td><td rowspan="4">$\mathsf{false}(\mathcal{I}_i) \cap \mathsf{false}(\mathcal{I}_j) = \emptyset$</td><td>10</td></tr>
<tr><td>$(i < j) \wedge \mathsf{goodCandidate}(\mathcal{I}_i, \mathcal{I}_j)$</td><td>9</td></tr>
<tr><td>$\mathcal{I}_j$ is removed when $\mathcal{I}_i$ is replaced</td><td rowspan="2">11</td></tr>
<tr><td>$\mathcal{I}_i$ and $\mathcal{I}_j$ are removed when another interpretation is replaced</td></tr>
<tr><td>$\exists c : (\mathcal{I} \not\models c) \wedge (\mathcal{I}_i$ covers $c) \wedge (\mathcal{I} \not\models \mathcal{H}_i)$</td><td>$\mathsf{true}(\mathcal{I}_i) \not\subseteq \mathsf{true}(\mathcal{I})$</td><td>5</td></tr>
<tr><td>$\exists c : (\mathcal{I}_j \not\models c) \wedge (\mathcal{I}_i$ covers $c)$</td><td rowspan="3">$\mathsf{true}(\mathcal{I}_i) \not\subseteq \mathsf{true}(\mathcal{I}_j)$</td><td>5, 6</td></tr>
<tr><td>$\mathcal{I}_j$ is removed when $\mathcal{I}_i$ is replaced</td><td rowspan="2">11</td></tr>
<tr><td>$\mathcal{I}_i$ and $\mathcal{I}_j$ are removed when another interpretation is replaced</td></tr>
<tr><td rowspan="2">$\mathcal{J}$ is appended to $\mathfrak{L}$</td><td>$\neg\mathsf{goodCandidate}(\mathcal{I}_j, \mathcal{J})$</td><td rowspan="3">9</td></tr>
<tr><td>$\neg\mathsf{goodCandidate}(\mathcal{J}, \mathcal{I}_j)$</td></tr>
<tr><td>$(i < j) \wedge (\mathsf{false}(\mathcal{I}_i) \cap \mathsf{false}(\mathcal{I}_j) \neq \emptyset)$</td><td>$\neg\mathsf{goodCandidate}(\mathcal{I}_i, \mathcal{I}_j)$</td></tr>
</table>

Table 1: Summary of technical results (where $c \in \mathcal{T}$).

We present a polynomial upper bound on the number of iterations of the main loop via a bound function. That is, an expression that decreases on every iteration and is always ≥ 0 inside the loop body. Note that we obtain this upper bound even though the learner does not know the size $|\mathcal{T}|$ of the target. Table 1 presents a survey of some technical results used along the paper.

Lemma 12 *Let N be $2 \cdot |V|^2 \cdot |\mathcal{T}|$. The expression $E = |\mathfrak{L}| + (N - 2 \cdot \sum_{\mathcal{I} \in \mathfrak{L}} |\mathsf{false}(\mathcal{I})|)$ always evaluates to a natural number inside the loop body and* *decreases on every iteration.*

Proof. By Corollary 1, the size of $\mathfrak{L}$ is bounded at all times by $|V| \cdot |\mathcal{T}|$. Thus, N is an upper bound for $2 \cdot \sum_{\mathcal{I} \in \mathfrak{L}} |\mathsf{false}(\mathcal{I})|$, which means that E always evaluates to a natural number. It remains to show that E decreases on every iteration. In each iteration there are three possibilities: (1) an element $\mathcal{I}$ is appended. Then, $|\mathfrak{L}|$ increases by one but $|\mathsf{false}(\mathcal{I})| \geq 1$ 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 11 of Algorithm 2. By Point 2 of Lemma 11, 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 11, 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($\mathcal{J}$)). 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 12, the total number of iterations of Algorithm 2 is bounded by a polynomial in $|\mathcal{T}|$ and $|V|$. We now state our main result.

Theorem 2 *The problem of learning MVDF from* 2*-quasi-Horn, more precisely the learning framework* $\mathfrak{F}_M$*, is polynomial time exact learnable.*

5. An Example Run

We describe an example run of Algorithm 2. Suppose the target MVDF is

$$\mathcal{T} = \{12 \to 345 \lor 678, 23 \to 146 \lor 578, 12678 \to 3 \lor 45\}.$$

Initially the sequence $\mathfrak{L}$ of interpretations and the hypothesis $\mathcal{H}$ are both empty. Suppose that the counterexample to our first equivalence query with $\mathcal{H}$ is

$12 \to 3 \vee 6$. Since $\mathcal{H}$ is empty, the return of the function 'SemanticTree' contains the interpretation $\mathcal{I}_1$ with $\mathsf{true}(\mathcal{I}_1) = \{1, 2\}$. Algorithm 2 appends $\mathcal{I}_1$ to $\mathfrak{L}$ and constructs the hypothesis $\mathcal{H}$. Then,

$$\mathfrak{L} = \{\mathcal{I}_1\} \text{ and } \mathcal{H} = \{12 \to 345 \vee 678\}.$$

Suppose that the counterexample to our second equivalence query with $\mathcal{H}$ is $12347 \to 6 \vee 8$. The return of the function 'SemanticTree' contains the interpretations $\mathcal{I}_2$ and $\mathcal{J}_2$, where $\mathsf{true}(\mathcal{I}_2) = \{1, 2, 3, 4, 5, 7\}$ and $\mathsf{true}(\mathcal{J}_2) = \{1, 2, 3, 4, 6, 7, 8\}$. Only $\mathcal{I}_2$ is a countermodel and, so, Algorithm 2 chooses $\mathcal{I}_2$ in Line 6. In the function 'RefineCounterModel', $(\mathcal{I}_2, \mathcal{I}_1)$ is not a $\mathsf{goodCandidate}$ because $\mathcal{I}_2 \cap \mathcal{I}_1 \not\models \mathcal{H}$ (it violates Point (ii) of Definition 1). For the same reason, in Line 8, $(\mathcal{I}_1, \mathcal{I}_2)$ is not a $\mathsf{goodCandidate}$ and, thus, Algorithm 2 appends $\mathcal{I}_2$ to $\mathfrak{L}$. The constructed sequence $\mathfrak{L}$ and hypothesis $\mathcal{H}$ are as follows:

$$\mathfrak{L} = \{\mathcal{I}_1, \mathcal{I}_2\} \text{ and } \mathcal{H} = \{12 \to 345 \vee 678, 123457 \to 6 \vee 8\}.$$

Suppose that the counterexample to our third equivalence query with $\mathcal{H}$ is $123458 \to 6 \vee 7$. The return of the function 'SemanticTree' contains the countermodel $\mathcal{I}_3$ with $\mathsf{true}(\mathcal{I}_3) = \{1, 2, 3, 4, 5, 8\}$ which is chosen in Line 6. Algorithm 2, then, calls the function 'RefineCounterModel' with $\mathcal{I}_3$ as input. We have that $(\mathcal{I}_3, \mathcal{I}_1)$ is not a $\mathsf{goodCandidate}$ (it violates Point (ii) of Definition 1) but $(\mathcal{I}_3, \mathcal{I}_2)$ *is* a $\mathsf{goodCandidate}$. Algorithm 2 returns from the function 'RefineCounterModel' with $\mathcal{I}_{3'} = \mathcal{I}_3 \cap \mathcal{I}_2$, i.e., $\mathsf{true}(\mathcal{I}_{3'}) = \{1, 2, 3, 4, 5\}$. Algorithm 2 replaces $\mathcal{I}_2$ with $\mathcal{I}_{3'}$. The constructed sequence $\mathfrak{L}$ and hypothesis $\mathcal{H}$ are as follows:

$$\mathfrak{L} = \{\mathcal{I}_1, \mathcal{I}_{3'}\} \text{ and } \mathcal{H} = \{12 \to 345 \vee 678, 12345 \to 6 \vee 78\}.$$

Now, suppose that the counterexample to our fourth equivalence query with $\mathcal{H}$ is $123678 \to 4 \vee 5$. The return of the function 'SemanticTree' contains the countermodel $\mathcal{I}_4$ with $\mathsf{true}(\mathcal{I}_4) = \{1, 2, 3, 6, 7, 8\}$ which is chosen in Line 6. Algorithm 2, then, calls the function 'RefineCounterModel' with $\mathcal{I}_4$ as input. We have that $(\mathcal{I}_4, \mathcal{I}_1)$ and $(\mathcal{I}_4, \mathcal{I}_{3'})$ are not $\mathsf{goodCandidates}$ (Point (ii) of Definition 1 is violated). For the same reason, in Line 8, $(\mathcal{I}_1, \mathcal{I}_4)$ and $(\mathcal{I}_{3'}, \mathcal{I}_4)$ are not

$\mathsf{goodCandidate}$s and, thus, Algorithm 2 appends $\mathcal{I}_4$ to $\mathfrak{L}$. The constructed sequence $\mathfrak{L}$ and hypothesis $\mathcal{H}$ are as follows:

$$\mathfrak{L} = \{\mathcal{I}_1, \mathcal{I}_{3'}, \mathcal{I}_4\} \text{ and } \mathcal{H} = \{12 \to 345 \vee 678, 12345 \to 6 \vee 78, 123678 \to 4 \vee 5\}.$$

Note that the last two mvds in $\mathcal{H}$ violate the second mvd in $\mathcal{T}$. However, if we omit Point (ii) of Definition 1, the oracle could give $12678 \to 3 \vee 4$ as counterexample. Without Point (ii), this counterexample would replace the second element of our sequence of interpretations with $\mathcal{J}$ such that $\mathsf{true}(\mathcal{J}) = \{1, 2\}$ (just like the first element of the sequence), the algorithm would enter into a loop.

Suppose that the counterexample to our fifth equivalence query with $\mathcal{H}$ is $23 \to 6 \vee 7$. In Line 6 we chose the interpretation $\mathcal{I}_5$ with $\mathsf{true}(\mathcal{I}_5) = \{2, 3\}$. The return of the function 'RefineCounterModel' is $\mathcal{I}_5$. In Line 8, $(\mathcal{I}_{3'}, \mathcal{I}_5)$ is a $\mathsf{goodCandidate}$ and, thus, Algorithm 2 replaces $\mathcal{I}_{3'}$ with $\mathcal{I}_5$. In Line 11, Algorithm 2 removes $\mathcal{I}_4$ from $\mathfrak{L}$. The constructed sequence $\mathfrak{L}$ and hypothesis $\mathcal{H}$ are as follows:

$$\mathfrak{L} = \{\mathcal{I}_1, \mathcal{I}_5\} \text{ and } \mathcal{H} = \{12 \to 345 \vee 678, 23 \to 146 \vee 578\}.$$

To finish our run, suppose that the counterexample to our sixth equivalence query is $12678 \to 3 \vee 4$. The interpretation $\mathcal{I}_6$ with $\mathsf{true}(\mathcal{I}_6) = \{1, 2, 6, 7, 8\}$ is appended to $\mathfrak{L}$ and Algorithm 2 constructs $\mathcal{H}$ equivalent to $\mathcal{T}$.

6. Conclusions and Open Problems

We presented an algorithm that exactly learns the class MVDF in polynomial time from 2-quasi-Horn clauses. As this class is a generalization of Horn and a restriction of 2-quasi-Horn, we extended the boundary between 1 and 2-quasi-Horn of what can be efficiently learned in the exact model. We would like to find similar algorithms where the examples are either mvd clauses or interpretations. A more general open problem is whether the ideas presented here can be extended to handle other restrictions of 2-quasi-Horn. Another direction is to use our algorithm to develop software to support the design of database schemas in 4NF.

Acknowledgements

We would like to thank an anonymous referee for pointing out that the proof of Lemma 9 was incomplete in an earlier version and for giving many suggestions to improve the readability of the paper. Hermo was supported by the Spanish Project TIN2013-46181-C2-2-R, the Basque Project GIU12/26 and grant UFI11/45. Ozaki was supported by the Science without Borders scholarship programme.

References

[1] E. F. Codd, A relational model of data for large shared data banks, Communications of the ACM 13 (6) (1970) 377–387. doi:10.1145/362384.362685. URL http://doi.acm.org/10.1145/362384.362685

[2] C. Delobel, Normalization and hierarchical dependencies in the relational data model, ACM Transactions on Database Systems 3 (3) (1978) 201–222. doi:10.1145/320263.320271. URL http://doi.acm.org/10.1145/320263.320271

[3] R. Fagin, Multivalued dependencies and a new normal form for relational databases, ACM Transactions on Database Systems 2 (1977) 262–278.

[4] R. Khardon, H. Mannila, D. Roth, Reasoning with Examples: Propositional Formulae and Database Dependencies, Acta Informatica 36 (4) (1999) 267–286. URL http://l2r.cs.uiuc.edu/~{}danr/Papers/db.pdf

[5] Y. Sagiv, C. Delobel, D. S. Parker, Jr., R. Fagin, An equivalence between relational database dependencies and a fragment of propositional logic, Journal of the ACM 28 (3) (1981) 435–453. doi:10.1145/322261.322263. URL http://doi.acm.org/10.1145/322261.322263

[6] J. L. Balcázar, J. Baixeries, Characterizations of multivalued dependencies and related expressions, in: Discovery Science, 7th International Conference, DS 2004, Padova, Italy, October 2-5, 2004, Proceedings, 2004, pp. 306–313.

doi:10.1007/978-3-540-30214-8_25.
URL http://dx.doi.org/10.1007/978-3-540-30214-8_25

[7] D. Angluin, M. Frazier, L. Pitt, Learning conjunctions of Horn clauses, Machine Learning 9 (2) (1992) 147–164.

[8] M. Frazier, L. Pitt, Learning from entailment: An application to propositional Horn sentences, in: Machine Learning, Proceedings of the Tenth International Conference, University of Massachusetts, Amherst, MA, USA, June 27-29, 1993, 1993, pp. 120–127.

[9] M. Hermo, V. Lavín, Learning minimal covers of functional dependencies with queries, in: Proceedings of the 10th International Conference on Algorithmic Learning Theory, ALT'99, Springer-Verlag, 1999, pp. 291–300.
URL http://dl.acm.org/citation.cfm?id=647717.735778

[10] D. Angluin, Negative results for equivalence queries, Machine Learning 5 (2) (1990) 121–150. doi:10.1007/BF00116034.
URL http://dx.doi.org/10.1007/BF00116034

[11] V. Lavín, M. Hermo, Negative results on learning multivalued dependencies with queries, Information Processing Letters 111 (19) (2011) 968–972. doi:10.1016/j.ipl.2011.07.007.
URL http://dx.doi.org/10.1016/j.ipl.2011.07.007

[12] V. Lavín, On learning multivalued dependencies with queries, Theor. Comput. Sci. 412 (22) (2011) 2331–2339. doi:10.1016/j.tcs.2011.01.011.
URL http://dx.doi.org/10.1016/j.tcs.2011.01.011

[13] V. Lavín, Learning an extension of the class of functional dependencies with queries, New Generation Computing 33 (3) (2015) 319–340. doi:10.1007/s00354-015-0301-8.
URL http://dx.doi.org/10.1007/s00354-015-0301-8

[14] D. Angluin, Queries and concept learning, Mach. Learn. 2 (4) (1988) 319–342. doi:10.1023/A:1022821128753.
URL http://dx.doi.org/10.1023/A:1022821128753

[15] D. Angluin, Learning k-term dnf formulas using queries and counterexamples, Tech. rep., Department of Computer Science, Yale University (August 1987).

[16] K. Pillaipakkamnatt, V. Raghavan, Read-twice DNF formulas are properly learnable, Information and Computation 122 (2) (1995) 236 – 267. doi:http://dx.doi.org/10.1006/inco.1995.1149.
URL http://www.sciencedirect.com/science/article/pii/S0890540185711492

[17] M. D. Frazier, Matters horn and other features in the computational learning theory landscape: The notion of membership., Ph.D. thesis, University of Illinois Urbana, 36-37 (1994).