Learning Schema Mappings*

Balder ten Cate
UC Santa Cruz
btencate@ucsc.edu

Víctor Dalmau
Universitat Pompeu Fabra
victor.dalmau@upf.edu

Phokion G. Kolaitis
UC Santa Cruz &
IBM Research–Almaden
kolaitis@cs.ucsc.edu

ABSTRACT

A schema mapping is a high-level specification of the relationship between a source schema and a target schema. Recently, a line of research has emerged that aims at deriving schema mappings automatically or semi-automatically with the help of data examples, i.e., pairs consisting of a source instance and a target instance that depict, in some precise sense, the intended behavior of the schema mapping. Several different uses of data examples for deriving, refining, or illustrating a schema mapping have already been proposed and studied.

In this paper, we use the lens of computational learning theory to systematically investigate the problem of obtaining algorithmically a schema mapping from data examples. Our aim is to leverage the rich body of work on learning theory in order to develop a framework for exploring the power and the limitations of the various algorithmic methods for obtaining schema mappings from data examples. We focus on GAV schema mappings, that is, schema mappings specified by GAV (Global-As-View) constraints. GAV constraints are the most basic and the most widely supported language for specifying schema mappings.

We present an efficient algorithm for learning GAV schema mappings using Angluin's model of exact learning with membership and equivalence queries. This is optimal, since we show that neither membership queries nor equivalence queries suffice, unless the source schema consists of unary relations only. We also obtain results concerning the learnability of schema mappings in the context of Valiant's well known PAC (Probably-Approximately-Correct) learning model. Finally, as a byproduct of our work, we show that there is no efficient algorithm for approximating the shortest GAV schema mapping fitting a given set of examples, unless the source schema consists of unary relations only.

Categories and Subject Descriptors
H.2.5 [**Heterogeneous Databases**]: Data translation
I.2.6 [**Learning**]: Concept learning

General Terms
Languages, Algorithms, Theory

Keywords
Schema mappings, data exchange, computational learning theory

*This research was partially supported by NSF Grant IIS-0905276. We are grateful to Vitaly Feldman and David Helmbold for helpful discussions.

ICDT 2012, March 26–30, 2012, Berlin, Germany.

1. INTRODUCTION

A schema mapping is a high-level specification of the relationship between two database schemas, called the source schema and the target schema. Schema mappings constitute the basic building blocks in both data exchange and data integration (see the surveys [10, 35, 37]). Obtaining a schema mapping between two schemas is an indispensable first step in data exchange, data integration, and other major data inter-operability tasks.

Several different systems for designing and deriving schema mappings have already been built; these include Clio [29], HePToX [13], and Microsoft's mapping composer [11]. A common characteristic of these systems is that they produce schema mappings from visual annotations (e.g., attribute correspondences) between the source schema and the target schema. In turn, these visual annotations are obtained automatically or semi-automatically using schema matching techniques or via an interaction with the users.

In a parallel direction, researchers have investigated how schema mappings can be illustrated, refined, or derived using data examples, where a data example is a pair consisting of a source instance and a target instance. Papers in which data examples are used to illustrate and refine schema mappings include [2, 43], while papers in which data examples are used to derive schema mappings include [19, 26, 27, 4]. In particular, [27] derives a schema mapping from a single data example using a cost model that takes into account several different parameters, including the size of the syntactic description of the schema mapping. In general, given a set of data examples, there may exist several pairwise logically inequivalent schema mappings that "fit" the given data examples. In [4], an algorithm was designed that, given a finite set of data examples, decides whether or not there exists a GLAV schema mapping (i.e., a schema mapping specified by Global-and-Local-As-View constraints) that "fits" these data examples. If such a fitting GLAV schema mapping exists, then the algorithm returns the "most general" one such schema mapping. Similar results are obtained in [4] for the case of GAV schema mappings.

A related investigation has focused on the problem of which schema mappings can be uniquely characterized via a finite set of data examples [3, 40].

In this paper, we cast the problem of obtaining algorithmically a schema mapping from data examples as a learning problem and then embark on a systematic investigation of this problem using concepts and methods of computational learning theory. Our aim is to leverage the rich body of work on learning theory in order to

develop a framework for exploring the power and the limitations of the various algorithmic methods for obtaining schema mappings from data examples. Computational learning theory can be succinctly described as the mathematical analysis of machine learning problems and algorithms. It was pioneered by L. Valiant in his 1984 seminal paper [41], in which he proposed the efficient *Probably-Approximately-Correct* (PAC) learning model. Since that time, several other learning models have been proposed and investigated in depth. A particularly prominent and well-studied model is the *exact learning* model, which was introduced by D. Angluin [5].

We focus on GAV schema mappings, that is, schema mappings specified by *Global-As-View* (GAV) constraints. As is well known, GAV constraints form the most basic high-level language for specifying schema mappings; furthermore, GAV constraints are supported by essentially every information integration system, be it an industrial tool or an academic prototype. We will write $\mathrm{GAV}(\mathbf{S}, \mathbf{T})$ to denote the class of all GAV schema mappings over a given source schema $\mathbf{S}$ and target schema $\mathbf{T}$.

Here, we study the learnability of GAV schema mappings in Angluin's exact learning model. In this model, a learning algorithm has to identify a "goal" concept from a class of concepts by asking a number of queries that are answered by oracles; the main types of queries considered in the literature are *membership queries* and *equivalence queries*. In our context, the "goal" concept is some schema mapping that we wish to identify up to logical equivalence. A membership query asks whether a particular target instance is a solution for a particular source instance, while an equivalence query asks whether a particular schema mapping is logically equivalent to the goal schema mapping. One could also consider other types of queries, such as whether a particular target instance is a universal solution for a particular source instance, or asking for a universal solution for a particular source instance; however, it turns out that, in our context, both these types of queries are equivalent to membership queries. We say that a concept class is *efficiently exactly learnable with membership and equivalence queries* if there is a polynomial-time algorithm[1] such that for every goal schema mapping, the algorithm will, after asking a number of membership and equivalence queries, produce a candidate schema mapping that is logically equivalent to the goal schema mapping. The notion of *efficiently exactly learnable with membership queries* and the notion of *efficiently exactly learnable with equivalence queries* are defined in an analogous way by restricting the type of queries asked.

In the computational learning theory literature, the exact learning model with membership queries is regarded as suitable for learning problems involving reverse engineering, that is, for problems in which the specification of a "goal" device has to be discovered from the behavior of that device. In our context, exact learning with membership queries naturally captures a scenario in which we seek to discover the description of an underlying schema mapping from its data-exchange behavior. For this, the user may use data examples that are available in the form of a pair of source and target instances (e.g., entries for a particular author in the DBLP and the ACM digital library) or can directly construct small test data examples. The oracle for the membership problem could be the executable code for exchanging data according to some underlying schema mapping; the schema mapping itself may have been lost or, for proprietary reasons, only the executable code derived from that schema mapping is made available. Exact learnability with membership and equivalence queries can be naturally viewed as exact learning with membership queries, where, in addition, the learning algorithm is allowed to make a number of mistakes until it finds the "goal" concept. Intuitively, efficient exact learnability with membership and equivalence learning amounts to efficient learning with membership queries and a bounded number of mistakes.

Our main results yield a complete picture for the efficient learnability of GAV schema mappings in the exact learning model with membership and/or equivalence queries.

THEOREM A. *For every source schema $\mathbf{S}$ and every target schema $\mathbf{T}$, the concept class $\mathrm{GAV}(\mathbf{S}, \mathbf{T})$ of all GAV schema mappings from $\mathbf{S}$ to $\mathbf{T}$ is efficiently exactly learnable with membership and equivalence queries.*

The preceding Theorem A is optimal, in the sense that both membership and equivalence queries are needed, unless the source schema contains unary relation symbols only.

THEOREM B. *Let $\mathbf{S}$ be a source schema and $\mathbf{T}$ a target schema.*

1. *If $\mathbf{S}$ contains a relation symbol of arity at least two, then*
   a. *$\mathrm{GAV}(\mathbf{S}, \mathbf{T})$ is not efficiently exactly learnable with membership queries;*
   b. *$\mathrm{GAV}(\mathbf{S}, \mathbf{T})$ is not efficiently exactly learnable with equivalence queries,*
2. *If $\mathbf{S}$ contains only unary relation symbols, then*
   a. *$\mathrm{GAV}(\mathbf{S}, \mathbf{T})$ is efficiently exactly learnable with membership queries;*
   b. *$\mathrm{GAV}(\mathbf{S}, \mathbf{T})$ is efficiently exactly learnable with equivalence queries,*

Besides exact learnability with membership queries and/or equivalence queries, we also consider Valiant's PAC learning model [41]. In this model, roughly speaking, a learning algorithm is presented with a number of randomly generated examples, according to some probability distribution, where the examples are labeled according to an unknown target GAV schema mapping, and the algorithm has to produce a GAV schema mapping such that, on examples generated according to the same probability distribution, the hypothesis is approximately correct.

THEOREM C. *Let $\mathbf{S}$ be a source schema and $\mathbf{T}$ a target schema.*

1. *If $\mathbf{S}$ contains a relation symbol of arity at least two, then $\mathrm{GAV}(\mathbf{S}, \mathbf{T})$ is not efficiently PAC learnable, provided* RP $\neq$ NP *(where* RP *is randomized polynomial time).*
2. *If $\mathbf{S}$ contains only unary relation symbols, then $\mathrm{GAV}(\mathbf{S}, \mathbf{T})$ is efficiently PAC learnable.*

As a byproduct of our negative results about PAC learnability, we establish that, under standard complexity-theoretic assumptions, it is not possible to efficiently compute, given a set of data examples, a fitting GAV schema mapping whose size is within a polynomial of the size of the smallest fitting GAV schema mapping. This is a pure non-approximability result that is of interest in its own right (especially in the light of the work in [27]), but is obtained via our results on non-efficient PAC learnability.

In [40], the class $\mathrm{GAV\text{-}CA}(\mathbf{S}, \mathbf{T})$ of all *c-acyclic* GAV schema mappings was introduced. The main finding was that the c-acyclic GAV schema mappings are precisely those GAV schema mappings (up to logical equivalence) that are uniquely characterizable by a finite set of universal examples. In particular, this implies that the class $\mathrm{GAV\text{-}CA}(\mathbf{S}, \mathbf{T})$ of c-acyclic GAV schema mappings is learnable (but not necessarily efficiently learnable) with membership queries. Nevertheless, we show that the negative results for

[1]The precise definition of a polynomial-time algorithm in the context of exact learning is given in Section 2.2.

efficient learnability listed in Theorems B and C remain true when $\mathrm{GAV}(\mathbf{S}, \mathbf{T})$ is replaced by $\mathrm{GAV\text{-}CA}(\mathbf{S}, \mathbf{T})$.

Over the years, there have been numerous uses of machine learning in data management. For example, machine learning algorithms are ubiquitous in data mining. Furthermore, machine learning algorithms have been used to discover database dependencies (e.g., see [23, 25]) and database queries [32, 42, 30]. See also Section 2.5. In information integration, there has been a series of investigations on machine learning algorithms for schema matching, including [20, 21, 38]. To the best of our knowledge, the work reported here is the first systematic effort to analyze a problem in information integration from the viewpoint of computational learning theory.

2. PRELIMINARIES

2.1 Databases and Schema Mappings

Schemas, Instances, and Homomorphisms A *schema* is a nonempty finite sequence of relation symbols $\mathbf{S} = (S_1, \ldots, S_n)$, where each S_i has a positive integer $\mathsf{arity}(S_i)$ as its associated arity. An *instance* of the schema $\mathbf{S}$, or $\mathbf{S}$-instance, is a sequence $I = (S_1^I, \ldots, S_n^I)$, where each S_i^I is a relation (over some domain) of arity $\mathsf{arity}(S_i)$. If I is a database instance and $\mathbf{a}$ is a tuple belonging to the relation S_i^I, then we call $S_i(\mathbf{a})$ a *fact* of I. The *active domain* of a database instance I, denoted by $adom(I)$, is the collection of all values occurring in facts of I. If I and I' are instances of the same schema, then a *homomorphism* $h : I \to I'$ is a function from $adom(I)$ to $adom(I')$, such that for every fact $S_i(a_1, \ldots, a_k)$ of I, we have that $S_i(h(a_1), \ldots, h(a_k))$ is a fact of I'. If $\psi(\mathbf{x})$ is a conjunction of atomic formulas over S, then the *canonical database for* ψ is the $\mathbf{S}$-instance obtained by viewing every atomic formula of ψ as a fact.

The *direct product* $I \times I'$ of two $\mathbf{S}$-instances I, I', is the $\mathbf{S}$-instance whose active domain consists of pairs $\langle a, a' \rangle$ with $a \in adom(I)$ and $a' \in adom(I')$ and that contains all facts $S_i(\langle a_1, a_1' \rangle, \ldots, \langle a_k, a_k' \rangle)$ (with $k = \mathsf{arity}(S_i)$) such that $(a_1, \ldots, a_k) \in S_i^I$ and $(a_1', \ldots, a_k') \in S_i^{I'}$. The *disjoint union* $I \oplus I'$ of two $\mathbf{S}$ instance I, I' is the (unique-up-to-isomorphism) instance consisting of all facts that belong either to I or to I^*, where I^* is some isomorphic copy of I', such that I and I^* have disjoint active domains. For those readers familiar with universal algebra, direct product and disjoint union are precisely the meet and join operators of the homomorphism-lattice of $\mathbf{S}$-instances. An instance I is *connected* if it cannot be represented as the disjoint union of two non-empty $\mathbf{S}$-instances. Every instance can be represented as the disjoint union of its maximal connected subinstances, also called *connected components*.

Schema Mappings, Data Exchange and Universal Solutions A schema mapping is a declarative specification of the relationship between two database schemas. Formally, a schema mapping is a triple $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$, where $\mathbf{S}$ and $\mathbf{T}$ are database schemas, called the *source schema* and the *target schema*, and Σ is a finite set of constraints in the form of formulas of some logical language. Here, we will consider the logical language of *Global-As-View* (GAV) constraints, i.e., first-order sentence of the form

$$\forall \mathbf{x}(\phi(\mathbf{x}) \to \psi(\mathbf{x})),$$

where $\mathbf{x}$ is a sequence of variables, $\phi(\mathbf{x})$ is a conjunction of atomic formulas over $\mathbf{S}$, and $\psi(\mathbf{x})$ is an atomic formula over $\mathbf{T}$ such that each variable $x \in \mathbf{x}$ occurs in ϕ. By an *atomic formula* over a schema, here, we mean a formula of the form $R(x_1, \ldots, x_n)$ (repeating variables is allowed), where R belongs to the schema and has arity n. For example, the first-order sentence

$$\forall x, y, z(E(x, y) \wedge F(y, z) \to P(x, z))$$

is a GAV constraint. A *GAV schema mapping* is a schema mapping specified by a finite set of GAV constraints.

We will usually assume a fixed source schema $\mathbf{S}$ and target schema $\mathbf{T}$. By a *source instance*, we will mean an instance for the schema $\mathbf{S}$; by a *target instance*, we will mean an instance for the target schema $\mathbf{T}$. We will make a habit to write $I, I', \ldots$ for the source instances and to write $J, J', \ldots$ for the target instances. We say that a target instance J is a *solution* for a source instance I with respect to (w.r.t.) a schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ if $(I, J) \models \Sigma$, i.e., if I and J together satisfy all the constraints of $\mathcal{M}$. Two schema mappings $\mathcal{M}, \mathcal{M}'$ over the same schemas are *logically equivalent*, denoted by $\mathcal{M} \equiv \mathcal{M}'$, if for all source instances I and target instances J, J is a solution for I w.r.t. $\mathcal{M}$ if and only if J is a solution for I w.r.t. $\mathcal{M}'$.

Data exchange is the problem of constructing a solution for a given source instance with respect to a schema mapping. A source instance can have many solutions, typically infinitely many. *Universal* solutions have been identified as the preferred solutions for data exchange [22]. A solution J for a source instance I with respect to a schema mapping $\mathcal{M}$ is *universal* if for every other solution J' of I w.r.t. $\mathcal{M}$, there is a homomorphism $h : J \to J'$ that is constant on the active domain of I, i.e., such that $h(a) = a$ for all $a \in adom(I) \cap adom(J)$. Intuitively, a universal solution contains only information that necessarily belongs to a solution of the given source instance. For example, if $\mathcal{M}$ is the schema mapping specified by the above GAV constraint and I is the source instance consisting of the facts $E(a, b), E(a, d), F(b, c)$, then the target instance J consisting of just the fact $P(a, c)$ is a universal solution for I w.r.t. $\mathcal{M}$; in contrast, the target instance J' consisting of the facts $P(a, c)$ and $P(a, d)$ is a solution for I w.r.t. $\mathcal{M}$, but not a universal one. If $\mathcal{M}$ is a GAV schema mapping, then for every source instance I, there is a unique target instance J with $adom(J) \subseteq adom(I)$ that is a universal solution of I w.r.t. $\mathcal{M}$. This target instance, better known as the *canonical universal solution of* I and denoted by $\mathsf{can\text{-}sol}_{\mathcal{M}}(I)$, is computable from I in polynomial time [22].

Since we will work with schema mappings over a fixed source schema and target schema, we will often identify a schema mapping with its set of constraints.

2.2 Exact learning models

Informally, an *exact learning algorithm* is an algorithm that identifies an unknown goal concept by asking a number of queries about it. The queries are answered by an oracle that has access to the goal concept. In this paper, we consider the two most extensively studied kinds of queries: *membership* queries and *equivalence* queries. We will first review basic notions from computational learning theory, such as the notion of a *concept*, and then explain what it means for a concept class to be *efficiently exactly learnable with membership and/or equivalence queries*. Afterwards, we will explain how GAV schema mappings can be viewed as a concept class.

Let X be a (possibly infinite) set of *examples*. A *labeled* example is any member of $X \times \{0, 1\}$. We will refer to an example of the form $(x, 0)$ as a *negative example* and to an example of the form $(x, 1)$ as a *positive example*. A *concept over* X is a function $c : X \to \{0, 1\}$, and a *concept class* $\mathcal{C}$ is a collection of concepts. We will often implicitly identify a concept c with the set of all examples x such that $c(x) = 1$. It is assumed that concepts are specified using some representation system so that one can speak of the length of the specification of a concept. More formally, a *represen-*

tation system for $\mathcal{C}$ is a string language $\mathcal{L}$ over some finite alphabet Σ, together with a surjective function $r : \mathcal{L} \to \mathcal{C}$. For every concept representation $\ell \in \mathcal{L}$, we write $|\ell|$ to denote its length (as a string). For every concept $c \in \mathcal{C}$, we write size(c) to denote the length of the smallest representation, i.e., $\min_{r(\ell)=c} |\ell|$. Similarly, we assume a representation system, with a corresponding notion of length, for the examples in X. Sometimes, when there is no risk of confusion, we may forget about the distinction between a concept and its representation.

As an example encountered very often in computational learning theory, suppose that $X = \{0,1\}^n$ is the set of all n-ary boolean assignments. Then one might consider the concept class consisting of all Boolean functions $c : \{0,1\}^n \to \{0,1\}$ represented by means of DNFs (i.e., Boolean formulas in disjunctive normal form) over the set of variables $x_1, \ldots, x_n$.

For every concept c, we denote by MEM$_c$ the *membership oracle* for c, that is, the oracle that takes as input an example x and returns its label, $c(x)$, according to c. Similarly, for every concept $c \in \mathcal{C}$, we denote by EQ$_c$, the *equivalence oracle* for c, that is, the oracle that takes as input the representation of a concept h and returns "yes", if $h = c$, or a counterexample x in the symmetric difference $h \oplus c$, otherwise.

An *exact learning algorithm with membership and/or equivalence queries* for a concept class C is an algorithm `alg` that takes as input a natural number n and has access to the membership oracle and/or equivalence oracle for an unknown *goal concept*[2] $c \in C$ of size[3] at most n. The algorithm `alg` must terminate after finite amount of time and output (some representation of) the goal concept c. This notion was introduced by Angluin [5]. A little later on, Angluin [6] also introduced the notion of a *polynomial-time* exact learning algorithm. Specifically, we say that an exact learning algorithm `alg` with membership and/or equivalence queries *runs in polynomial time* if there exists a two-variable polynomial $p(n,m)$ such that at any point during the run of the algorithm, the time used by `alg` up to that point (counting one step per oracle call) is bounded by $p(n,m)$, where m the size of the largest counterexample returned by calls to the equivalence oracle up to that point in the run ($m = 0$ if no equivalence queries have been used).

There is a delicate issue about this notion of polynomial time that we now discuss. One might be tempted to relax the previous definition by requiring merely that the total running time is bounded by $p(n,m)$. However, this change in the definition would give rise to a *wrong* notion of a polynomial-time algorithms in this context by way of a loophole in the definition. Indeed, under this change, one could design a learning algorithm that, in a first stage, identifies the goal hypothesis by (expensive) exhaustive search and that, once this is achieved, forces —by asking equivalence queries with appropriate modification of the goal concept— the equivalence oracle to return large counterexamples that would make up for the time spent during the exhaustive search phase.

We say that the algorithm `alg` *exactly learns a concept class* $\mathcal{C}$ *with membership and/or equivalence queries* if for all natural numbers n and for all concepts $c \in \mathcal{C}$ with size$(c) \leq n$, when `alg` is run on input n and with a membership oracle MEM$_c$ and/or an equivalence oracle EQ$_c$, it outputs a representation of c.

A concept class $\mathcal{C}$ is *efficiently exactly learnable with membership and/or equivalence queries* if there is an exact learning algorithm with membership and/or equivalence queries for $\mathcal{C}$ that runs in polynomial time.

[2]We use the term "goal concept" instead of the term "target concept" (the standard term in computational learning theory), because the latter clashes with the use of the term "target" for schema mappings.

[3]While standard in learning theory literature, the assumption that the learning algorithm knows the size of the goal concept in advance is not used in any of our positive results concerning learnability.

2.3 Approximate learning models

Next, we consider Valiant's well known Probably Approximately Correct (PAC) learning model [41]. The task of a PAC learning algorithm is to learn an (unknown) goal concept $c \in \mathcal{C}$, or, rather, an approximation of it, on the basis of labeled examples that have been randomly generated according to some probability distribution. Recall that $[0,1]$ is the closed unit interval of the real numbers, and recall that a *probability distribution* over X is a function $D : X \to [0,1]$ such that $\Sigma_{x \in X} D(x) = 1$. For every concept $c \in \mathcal{C}$ and probability distribution $D : X \to [0,1]$, we denote by $EX_{c,D}$ the *example oracle* for c and D, i.e., the oracle that, upon request, returns a labeled example $(x, c(x))$, where $x \in X$ is randomly chosen according to the probability distribution D. When we speak of an example oracle, we will mean the example oracle of some concept $c \in \mathcal{C}$ and some probability distribution $D : X \to [0,1]$.

A *PAC algorithm* is an algorithm `alg` that takes as input a rational accuracy parameter $0 < \epsilon < 1$, a rational confidence parameter $0 < \delta < 1$, and a natural number n, and that has access to an example oracle for an unknown goal concept of size at most n. The algorithm `alg` must terminate after a finite amount of time and output (a representation of) a concept $h \in C$. The output "hypothesis" concept h produced by `alg` should be seen as a guess of what the goal concept c may be. The *error*, error(h), for output concept h with respect to distribution D and goal concept c is

$$\text{error}(h) = \Pr_{x \in D}(c(x) \neq h(x))$$

Note that error$(\cdot)$ has an implicit dependence of c and D which we will omit for brevity when no confusion will result. We say that `alg` *learns a concept class* $\mathcal{C}$ if for all rational numbers ϵ and δ with $0 < \epsilon, \delta < 1$, for all natural numbers n, for all concepts c with size$(c) \leq n$, and all probability distributions D over the example set X of $\mathcal{C}$, when `alg` is run with inputs ϵ, δ, n and oracle $EX_{c,D}$, `alg` outputs with probability at least $1 - \delta$ a concept h with error$(h) \leq \epsilon$. The probability above is taken over the possible outputs of the random example oracle [9, 34].

Ideally, we would like that both the number of oracle calls and the computation cost are small. This is formalized by the following definition: `alg` *runs in polynomial time* if its running time (counting one step per oracle call) is bounded by a fixed polynomial in $1/\epsilon$, $1/\delta$, n, and the maximum size of a labeled example returned by a call to the oracle.

As usual in computational learning theory, we allow also randomized PAC algorithms, in other words, we allow for PAC algorithm that, besides the example oracle, also have access to a *coin-flip oracle*, which randomly generates 0 or 1 with equal probability.

A concept class $\mathcal{C}$ is *PAC learnable* if there is a PAC algorithm `alg` that learns it. If, furthermore, `alg` runs in polynomial time then we say that $\mathcal{C}$ is *efficiently PAC learnable*.

Exact learnability vs. PAC learnability. In [5], Angluin showed that, in many cases, exact learnability with equivalence queries implies PAC learnability. To state this result precisely, we need to first introduce some definitions. The *evaluation problem* for a concept class $\mathcal{C}$ asks: given the representation of a concept $c \in \mathcal{C}$ and the representation of an example x, does $x \in c$? We say that a concept class $\mathcal{C}$ is *polynomial-time evaluable* if there is a polynomial-time algorithm for the evaluation problem for $\mathcal{C}$. Angluin [5] showed that if $\mathcal{C}$ is a polynomial-time evaluable class that is efficiently exactly learnable with equivalence queries, then $\mathcal{C}$ is also efficiently

PAC learnable. Furthermore, this also holds if one adds membership queries to both sides. The vast majority of the learning literature deals with concept classes that are polynomial-time evaluable, such as, for example, the concept class of all DNF formulas. In contrast, as we will discuss later on, here we are dealing with concept classes that are not polynomial-time evaluable. For this reason, we will need the following generalization that can be obtained by an straightforward adaptation of Angluin's proof.

THEOREM 2.1. (Direct adaptation of [5]) *If a concept class $\mathcal{C}$ is efficiently exactly learnable with equivalence queries (respectively, with equivalence and membership queries), then $\mathcal{C}$ is efficiently PAC learnable (respectively, PAC learnable with membership queries), provided the PAC algorithm has access to an oracle that solves the evaluation problem for $\mathcal{C}$.*

The class of DNF formulas is one of the most studied concept classes in computational learning theory, especially in the context of exact learning with queries. Angluin [5, 6] proved that DNFs are not efficiently learnable with membership or equivalence queries alone. Later on, it was shown in [31] that DNFs are not efficiently exactly learnable even if both membership and equivalence queries are allowed. It is also known that, assuming RP $\neq$ NP, DNFs are not learnable in the PAC model [1], even if we allow membership queries [24]. The fragment of monotone DNF, obtained by requiring every clause to contain only positive literals, has a different behavior. If only membership or only equivalence queries are allowed, then monotone DNF are not efficiently exactly learnable [5, 6]. In contrast, monotone DNF are efficiently learnable with equivalence and membership queries [5]; this yields an efficient PAC learning algorithm with membership queries for monotone DNF.

2.4 Schema mappings as a concept class

Fix a source schema $\mathbf{S}$ and a target schema $\mathbf{T}$. Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ be a GAV schema mapping (where Σ is a finite set of GAV constraints). A *data example* is a pair (I, J), where I is a source instance, i.e., an instance for $\mathbf{S}$, and J is a target instance, i.e., an instance for $\mathbf{T}$. A data example (I, J) is said to be a *positive example for* $\mathcal{M}$ if $(I, J) \models \Sigma$, in other words if it satisfies all constraints of $\mathcal{M}$; otherwise, it is said to be a *negative example for* $\mathcal{M}$. Semantically, we can identify a GAV schema mapping with the set of all its positive examples, and hence we can view it as a concept, where the example space is the set of all data examples. We shall denote by $\mathrm{GAV}(\mathbf{S}, \mathbf{T})$ the concept class thus defined.

It is worth noting that $\mathrm{GAV}(\mathbf{S}, \mathbf{T})$ has a different characteristic when compared with most concept classes that have been studied in computational learning theory literature, namely, it is not polynomial-time evaluable. Indeed, the following problem is coNP-complete: given a data example (I, J) and a GAV schema mapping $\mathcal{M}$, is (I, J) a positive or a negative example for $\mathcal{M}$?

Besides positive and negative examples, it is natural to consider also universal examples, i.e., data examples (I, J) such that J is a universal solution for I with respect to the schema mapping that is to be learned. We will come back to this in Section 6.2, and for the time being we will consider only positive and negative examples.

2.5 Related Work

A GAV constraint can be equivalently represented by a pair consisting of a target relation and a conjunctive query over the source schema (where the target relation and the query have the same arity), or, alternatively, can be viewed as a non-recursive Datalog rule. The PAC-learnability of the latter types of concepts has been studied in [16, 32, 42, 30]. The learning model considered in these papers differs from ours in several important ways.

The *simple instance uniform PAC-learnability* model used in [16], when recast in terms of our present terminology, can be paraphrased roughly as follows: the learning algorithm receives, as input, a single source instance I. The algorithm might request any number of random labelled examples, which are facts over the target schema labelled according to whether they do or do not belong to the universal solution, J^g, of I with respect to a goal schema mapping $\mathcal{M}^g$ (over a target schema consisting of a single relation). The algorithm is required to terminate and output a GAV schema mapping $\mathcal{M}^h$ such that the universal solution J^h, with respect to $\mathcal{M}^h$ is probably approximately equal to J^g, under the usual confidence and error conditions of the PAC model. Our learning model is more demanding, since we require the schema mapping produced by the algorithm to perform well on arbitrary source instances. The work in [32] considers, in addition, a variant of the previous model, called *extended instance uniform PAC-learnability*. This model was previously considered in the study of learnability of Prolog programs [17, 18].

The papers [16, 32] also consider the *PAC-prediction model* (also known as *improper PAC-learnability*). The PAC-prediction model is more relaxed in the sense that the hypothesis produced by the algorithm does not need to belong to the concept class (i.e., in our case, does not need to be a GAV schema mapping). However, in the PAC-prediction model, the output concept class is required to be polynomial-time evaluable; since GAV schema mappings are not polynomial-time evaluable, it follows that one cannot derive hardness of PAC-learning from hardness of PAC-prediction for the concept class of GAV schema mappings. In particular, the standard techniques for showing hardness in the context of the PAC-prediction model, as in [39, 8, 33], cannot be used directly in order to prove PAC hardness results for GAV schema mappings.

Other differences with the work we present here are that the results in [16, 32] concern almost exclusively the problem of learning (what corresponds to) a single GAV constraint and that in [16, 32] the target schema is not fixed but is treated as part of the input of the learning problem.

A simple learning algorithm for "polynomially generable" subclasses of the class of unions of conjunctive queries is presented and analyzed in [42]. Unions of conjunctive queries over a fixed finite schema consisting of unary relations only are an example of such a class. With minor modifications the algorithm given in [42] could therefore be adapted to learn, in the PAC model, GAV schema mappings over a unary source schema (Theorem C.2, cf. also Corollary 5.3).

In [30], the PAC learnability of conjunctive queries is studied. However, in order to circumvent the issue of non-polynomial time evaluability, a uniform bound is assumed on the number of elements of examples, and on the number of existentially quantified variables in the conjunctive queries.

GAV constraints can also be viewed as a special subclass of the first-order universal Horn formulas. The learnability of *propositional* Horn formulas has been investigated before in [7], where it was shown that this concept class is efficiently exactly learnable with membership and equivalence queries, while it is not efficiently learnable with membership queries alone, or with equivalence queries alone.

3. EFFICIENT EXACT LEARNABILITY WITH EQUIVALENCE AND MEMBERSHIP QUERIES

We shall design a polynomial time algorithm `alg` that identifies (up to logical equivalence) a goal GAV schema mapping $\mathcal{M}^g$ using

equivalence and membership queries. The algorithm `alg` works by maintaining an internal hypothesis, which is a GAV schema mapping $\mathcal{H}$ consisting of constraints that are *critically sound* with respect to the goal schema mapping $\mathcal{M}^g$. We say that a GAV constraint C is *critically sound* with respect to $\mathcal{M}^g$ if $\mathcal{M}^g \models C$ (i.e., C is logically implied by the GAV constraints of $\mathcal{M}^g$) and for every GAV constraint C' obtained by removing one of the conjuncts of the left-hand side of C, we have that $\mathcal{M}^g \not\models C'$.

In what follows, we will often identify a GAV constraint $\forall \mathbf{x}(\phi \rightarrow \psi)$ with the pair (I_ϕ, J_ψ) where I_ϕ, J_ψ are the canonical instances of ϕ and of ψ (observe that J_ψ must consist of a single fact). We also identify the GAV constraint $\forall \mathbf{x}(\phi \rightarrow \psi)$ with the $(\mathbf{S} \cup \mathbf{T})$-instance $I_\phi \cup J_\psi$. In this way, a homomorphism $h : C \rightarrow C'$ between GAV constraints is a function that maps atomic formulas occurring in the left-hand side or right-hand size of C to atomic formulas occurring in the left-hand-side or right-hand side of C'.

The following lemma is a direct generalization of a result by Chandra and Merlin [14].

LEMMA 3.1. *For all GAV constraints $C = (I, F)$, the following are equivalent:*

1. $\mathcal{M}^g \models C$
2. $C' \rightarrow C$ *for some* $C' \in \mathcal{M}^g$.
3. $F \in \mathsf{can\text{-}sol}_{\mathcal{M}^g}(I)$

LEMMA 3.2. *Given a source instance I, one can compute $\mathsf{can\text{-}sol}_{\mathcal{M}^g}(I)$ with at most $poly(|I|)$ many membership queries.*

PROOF. For each potential target fact F, ask a membership query to test if the target instance containing all possible facts except F is a solution to I. Finally, take $\mathsf{can\text{-}sol}_{\mathcal{M}^g}(I)$ to be the set of all facts for which the answer to the membership query is "no". □

LEMMA 3.3. *Given a GAV constraint C, one can test if $\mathcal{M}^g \models C$ with at most $poly(|C|)$ many membership queries.*

PROOF. Let $C = (I, F)$. It suffices to use Lemma 3.2 and test if $\mathsf{can\text{-}sol}_{\mathcal{M}^g}(I)$ contains F. □

LEMMA 3.4. *If $\mathcal{M}^g \models C$, then by asking at most $poly(|C|)$ many membership queries, one can compute in polynomial time a GAV constraint $\mathsf{crit}_{\mathcal{M}^g}(C)$ with the following properties:*

1. $\mathsf{crit}_{\mathcal{M}^g}(C) \subseteq C$ *(when $\mathsf{crit}_{\mathcal{M}^g}(C)$ and C are viewed as instances), and*
2. $\mathsf{crit}_{\mathcal{M}^g}(C)$ *is critically sound with respect to $\mathcal{M}^g$.*

PROOF. Let $C = (I, F)$, and let J be the target instance containing all possible facts (over the active domain of I) except for F. Clearly, (I, J) will be a negative example for $\mathcal{M}^g$. Repeatedly, try to remove facts from I while the example remains negative, until a minimal subinstance of I is reached. The source instance $I' \subseteq I$ obtained in this way is such that (I', F) is critically sound with respect to $\mathcal{M}^g$. □

For two GAV constraints C, C' that contain the same target relation, we denote by $C \times C'$ the GAV constraint that, viewed as an instance, is the direct product of C and C', if well defined. It may happen that $C \times C'$ is not well defined even if C and C' have the same target relation. For example, if C and C' are the GAV constraints $E(x, y) \rightarrow T(x)$ and $E(y, x) \rightarrow T(x)$ respectively, then their direct product, $E(\langle x, y\rangle, \langle y, x\rangle) \rightarrow T(\langle x, x\rangle)$, is not a well-defined GAV constraint as the variable $\langle x, x\rangle$ occurs in the right side but not in the left side.

However, in what follows, whenever we take a product of two GAV constraints, the result is well-defined, as will be guaranteed by the following lemma.

```
H := ∅
while the equivalence query H ≡ M^g fails do
    Let (I, J) be a counterexample to H ≡ M^g
    // As we will prove below, (I, J) is guaranteed to be a
    // positive example for H and a negative example for M^g
    // In particular, can-sol_{M^g}(I) ⊈ J
    Choose a fact F ∈ can-sol_{M^g}(I) \ J
    // Note that, by Lemma 3.1, M^g ⊨ (I, F)
    if there is a C ∈ H such that M^g ⊨ (I, F) × C, then
        Let C ∈ H be the most recently added GAV
            constraint for which it holds that M^g ⊨ (I, F) × C
        C' := (I, F) × C.
    else
        C' := (I, F)
    end if
    Add crit_{M^g}(C') to H   // using Lemma 3.4
end while
return(H)
```

Figure 1: Algorithm for learning GAV schema mappings using equivalence and membership queries

LEMMA 3.5. *Let C, C_1, C_2 be GAV constraints, such that there are homomorphisms $h_1 : C \rightarrow C_1$ and $h_2 : C \rightarrow C_2$. Then $C_1 \times C_2$ is well defined.*

PROOF. Let $T(x_1, \ldots, x_n)$ be the right-hand side of C. It follows that the right-hand side of C_1 is $T(h_1(x_1), \ldots, h_1(x_n))$ and the right-hand side of C_2 is $T(h_2(x_1), \ldots, h_2(x_n))$. Thus, the right-hand side of $C_1 \times C_2$ is $T(\langle h_1(x_1), h_2(x_1)\rangle, \ldots, \langle h_1(x_n), h_2(x_n)\rangle)$. Therefore, it suffices to show that each pair $\langle h_1(x_i), h_2(x_i)\rangle$ occurs in some source fact of $C_1 \times C_2$. This is indeed the case: since C is a well-defined GAV constraint, x_i occurs in some source tuple in C, and applying the homomorphisms h_1 and h_2 to this tuple yields a source tuple belonging to $C_1 \times C_2$ that contains $\langle h_1(x_i), h_2(x_i)\rangle$. □

The promised algorithm `alg` is now given in Figure 1.

Let us denote by $\mathcal{H}_i$ the value of the variable $\mathcal{H}$ after the i-th iteration of the **while** loop, where $\mathcal{H}_0 = \emptyset$, and $\mathcal{H}_i$ is undefined if i is larger than the total number of iterations of the **while** loop; let us also denote by C_i the GAV constraint that was added at the i-th iteration, i.e., $\mathcal{H}_i = \mathcal{H}_{i-1} \cup \{C_i\}$.

For every GAV schema mapping $\mathcal{H}$, and for every $C \in \mathcal{M}^g$, let $\mathcal{H}(C) = \{C' \in \mathcal{H} \mid C \rightarrow C'\}$.

LEMMA 3.6. *For every $i \geq 0$ such that $\mathcal{H}_i$ is defined, the following statements are true.*

1. *$\mathcal{H}_i$ consists of GAV constraints that are critically sound with respect to $\mathcal{M}^g$. In particular, a counterexample for the equivalence $\mathcal{H}_i \equiv \mathcal{M}^g$ must be a positive example for $\mathcal{H}_i$ and a negative example for $\mathcal{M}^g$.*
2. *If $i > 0$, then there is no $C \in \mathcal{H}_{i-1}$ such that $C \rightarrow C_i$. In other words, $\mathcal{H}_{i-1} \not\models C_i$.*
3. *If $C \in \mathcal{M}^g$, then the set $\mathcal{H}_i(C)$ is either empty or has a minimal element with respect to the homomorphism preorder $\rightarrow$.*

PROOF. (1) Follows directly from the definition of the algorithm, together with Lemma 3.1 and Lemma 3.4.

(2) Let I, J, F and C' be as computed in the i-th iteration of the algorithm. In particular, (I, J) is a counterexample to the equivalence $\mathcal{H}_{i-1} \equiv \mathcal{M}^g$. Since by the previous item we know that each constraint of $\mathcal{H}_{i-1}$ is sound with respect to $\mathcal{M}^g$, we know that (I, J) is a positive example for $\mathcal{H}_{i-1}$ and a negative example for $\mathcal{M}^g$. Since $F \notin J$, it follows that $\mathcal{H}_{i-1} \not\models (I, F)$, i.e., there is no $C \in \mathcal{H}_{i-1}$ such that $C \to (I, F)$ (cf. Lemma 3.1). By well known basic properties of direct products, it follows that there is no $C \in \mathcal{H}_{i-1}$ such that $C \to C''$; consequently, since $C_i = \mathsf{crit}_{\mathcal{M}^g}(C')$, which is a subinstance of C', there is no $C \in \mathcal{H}_{i-1}$ with $C \to C_i$.

(3) By induction on i. Let $C \in \mathcal{M}^g$. If $C_i \notin \mathcal{H}_i(C)$, then the result follows from the induction hypothesis. Therefore, let $C_i \in \mathcal{H}_i(C)$, i.e., $C \to C_i$. We will show that, in this case, C_i is a minimal element of $\mathcal{H}_i(C)$ with respect to the homomorphism preorder $\to$.

We distinguish two cases:

1. There is no $C_j \in \mathcal{H}_{i-1}$ such that $C \to C_j$. In this case, C_i is the only GAV constraint of $\mathcal{H}_i$ into which C maps and hence the result holds trivially.

2. There are $C_j \in \mathcal{H}_{i-1}$ such that $C \to C_j$. By induction hypothesis, we then know that there is such a $C_j \in \mathcal{H}_{i-1}$ that is minimal with respect to the homomorphism preorder, i.e., $C_j \to C_k$ for all $C_k \in \mathcal{H}_{i-1}$ that have the property that $C \to C_k$. In particular, by the previous item, we have that C_j is the most recently added GAV constraint with the property that $C \to C_j$. Now, let I, J, F, and C' be as in the i-th iteration of the algorithm. From the description of the algorithm and from the above remarks, it follows that $C_i = \mathsf{crit}_{\mathcal{M}^g}((I, F) \times C_k)$ for some C_k that was added no less recently than C_j. Since $C \to C_i$, we have that $C \to C_k$. Hence, C_j was added no less recently than C_k. In fact, this means that $C_j = C_k$ and hence $C_i = \mathsf{crit}_{\mathcal{M}^g}((I, F) \times C_j)$. It follows that $C_i \to C_j$ and, since, by item (2), $C_j \not\to C_i$, we can infer that C_i is a minimal element of $\mathcal{H}_i(C)$ with respect to the homomorphism preorder $\to$. This concludes the proof. □

THEOREM 3.7. *The algorithm* `alg` *terminates after at most* $|\mathcal{M}^g|$ *many iterations and returns a GAV schema mapping* $\mathcal{H}$ *that is logically equivalent to* $\mathcal{M}^g$.

PROOF. For every $n \geq 1$, define s_n to be $\sum_{C \in \mathcal{M}^g} s_n^C$, where s_n^C is 0 if $\mathcal{H}_n(C)$ is empty, and the number of variables occurring in the minimal element of $\mathcal{H}_n(C)$, otherwise (this is well defined by Lemma 3.6(3)). We claim that, for every $n \geq 1$, $s_{n+1} > s_n$. This shows that `alg` terminates after at most as many iterations as the total number of variables in the constraints of $\mathcal{M}^g$. That the schema mapping $\mathcal{H}$ returned is logically equivalent to $\mathcal{M}^g$ follows directly from the definition of the algorithm.

We now prove the claim. Assume that $\mathcal{H}_{n+1} = \mathcal{H}_n \cup \{C_{n+1}\}$. Since C_{n+1} is critically sound w.r.t $\mathcal{M}^g$, there is some $C \in T$ such that $C \to C_{n+1}$. By Lemma 3.6(2) and 3.6(3), C_{n+1} is the minimum of $\mathcal{H}_{n+1}(C)$. It follows that s_{n+1}^C is the domain size of C_{n+1}. If $\mathcal{H}_n(C) = \emptyset$, then there is nothing to prove. Otherwise, let C' be its minimal element. We know that $C_{n+1} \to C'$. In fact, the homomorphism in question must be surjective, in other words, C' must be the homomorphic image of C, for otherwise, we could obtain a non-surjective homomorphism from C to C' contradicting, via Lemma 3.1, the fact that C' is critically sound with respect to $\mathcal{M}^g$. We also know by Lemma 3.6(2) that C is not isomorphic to C'. It follows that the domain size of C_{n+1} is larger than that of C' and we are done. □

We have just proved the following result, which was stated as Theorem A in the Introduction.

THEOREM 3.8. *For every source schema* $\mathbf{S}$ *and every target schema* $\mathbf{T}$*, the concept class* $\mathrm{GAV}(\mathbf{S}, \mathbf{T})$ *of all GAV mappings from* $\mathbf{S}$ *to* $\mathbf{T}$ *is efficiently exactly learnable with equivalence and membership queries.*

By Theorem 2.1, an efficient algorithm with equivalence and membership queries can be transformed into a PAC algorithm with membership queries, provided the PAC algorithm has access to an oracle that solves the evaluation problem. This leads to the following result.

COROLLARY 3.9. *For every source schema* $\mathbf{S}$ *and every target schema* $\mathbf{T}$*, the concept class* $\mathrm{GAV}(\mathbf{S}, \mathbf{T})$ *is efficiently PAC learnable with membership queries and an oracle for* NP.

As we will see in Section 4, $\mathrm{GAV}(\mathbf{S}, \mathbf{T})$ is not efficiently PAC learnable in the standard PAC model without membership queries. We could not establish or refute the necessity of the NP oracle.

4. HARDNESS OF LEARNING RESULTS

We show that if the source schema $\mathbf{S}$ contains a relation of arity at least 2, then $\mathrm{GAV}(\mathbf{S}, \mathbf{T})$ is not efficiently exactly learnable with equivalence queries alone, is not efficiently exactly learnable with membership queries alone, and is not efficiently PAC learnable.

We first need to introduce some auxiliary results and definitions. An *oriented path* is a directed graph (digraph) that can be obtained from an undirected path by orienting each edge in exactly one way. For every $k \geq 1$, let G_k be the oriented path

$$\overrightarrow{3}\ \underbrace{\overleftarrow{1}\ \overrightarrow{2} \cdots \overleftarrow{1}\ \overrightarrow{2}}_{k \text{ times}}\ \overleftarrow{4},$$

that is, the oriented path consisting of 3 forward edges, followed by k segments each consisting of a backward edge and two forward edges, and ending with 4 backward edges.

We choose this family of digraphs because there are homomorphically incomparable. Formally, if we use the notation $G \to G'$ to denote the existence of a homomorphism from G to G', we have

LEMMA 4.1. *For all* $k, k' \geq 1$*, if* $k \neq k'$ *then* $\mathsf{G}_k \not\to \mathsf{G}_{k'}$.

We omit the proof of Lemma 4.1, which is straightforward (it uses the fact that every homomorphism $h : \mathsf{G}_k \to \mathsf{G}_{k'}$ has to send the sequence of 4 backward edge of G_k to the sequence of 4 backward edges of $\mathsf{G}_{k'}$).

Since the digraphs constructed are oriented paths, we have the following lemma.

LEMMA 4.2. *[28] There is a polynomial time algorithm that, given digraphs* A *and* G_k*,* $k \geq 1$*, decides correctly whether* $A \to \mathsf{G}_k$*. This implies also that it is possible to decide in polynomial time whether, given digraphs* $A, \mathsf{G}_k, \mathsf{G}_{k'}$*, we have that* $A \to \mathsf{G}_k \times \mathsf{G}_{k'}$.

Throughout this section, R will be a relation in $\mathbf{S}$ of arity $r \geq 2$ and T will be a relation in $\mathbf{T}$. The following construction is a technicality necessary to reduce to the case in which R has arity exactly 2. For every digraph $G = (V, E)$, let $\widehat{G}$ be the $\mathbf{S}$-instance containing for every $(x, y) \in E$, the fact $R(x, y, z_1, \ldots, z_{r-2})$, where $z_1, \ldots, z_{r-2}$ are fresh elements (which are different for every tuple). The following easy fact is used often in our proofs: for every digraphs G and G', $G \to G'$ if and only if $\widehat{G} \to \widehat{G'}$.

4.1 Hardness of efficient learning with membership queries

It was shown in [3] that there are GAV schema mappings that cannot be uniquely characterized by any finite set of data examples. In particular, if $\mathcal{M}$ is the schema mapping defined by the GAV constraint $\forall x, y(S(x) \wedge R(y, y) \rightarrow T(x))$, then for every finite set X of data examples, there is a schema mapping $\mathcal{M}'$ that is not logically equivalent to $\mathcal{M}$ and such that for each data example $(I, J) \in X$, (I, J) is a positive example for $\mathcal{M}'$ if and only if (I, J) is a positive example for $\mathcal{M}$. It follows that $\mathrm{GAV}(\mathbf{S}, \mathbf{T})$ is not exactly learnable by a learning algorithm that uses only membership queries and that does not know the size of the goal schema mapping in advance (since such a learning algorithm will never be able to establish with certainty that $\mathcal{M}$ is the goal schema mapping). The next theorem shows that $\mathrm{GAV}(\mathbf{S}, \mathbf{T})$ is not efficiently exactly learnable with membership queries, even when the learning algorithm knows the size of the goal schema mapping in advance (recall that we defined exactly learnability this way in Section 2). In the Introduction, this result was stated as Part 1a of Theorem B.

THEOREM 4.3. *Let* $\mathbf{S}$ *be a source schema and* $\mathbf{T}$ *a target schema. If* $\mathbf{S}$ *contains a relation symbol of arity at least two, then* $\mathrm{GAV}(\mathbf{S}, \mathbf{T})$ *is not efficiently exactly learnable with membership queries.*

PROOF. *(Sketch)* We adapt an argument used in the proof of Theorem 2 in [5]. Fix $N > 0$. For every N-ary boolean tuple (i.e., bitstring) $\mathbf{b} = (b_0, \ldots, b_{N-1}) \in \{0, 1\}^N$, we construct a GAV schema mapping $\mathcal{M}_{\mathbf{b}}$ containing the following GAV constraints:

- For every $0 \leq k < N$, a GAV constraint stating that, if $\widehat{\mathsf{G}_{2k}}$ and $\widehat{\mathsf{G}_{2k+1}}$ are homomorphic to the source instance, and the source instance contains a tuple $R(u_0, \ldots, u_{r-1})$, then $T(u_0 \ldots u_0)$ holds in the target instance.

  It is easy to write this GAV constraint using the canonical queries for $\widehat{\mathsf{G}_{2k}}$ and $\widehat{\mathsf{G}_{2k+1}}$. In particular, it can be written as

  $$\forall \mathbf{x_0}, \mathbf{x_1}, \mathbf{y}(q_{\widehat{\mathsf{G}_{2k}}}(\mathbf{x_0}) \wedge q_{\widehat{\mathsf{G}_{2k+1}}}(\mathbf{x_1}) \wedge R(\mathbf{y}) \rightarrow T(y_0 \ldots y_0)),$$

  where y_0 is the first variable in $\mathbf{y}$ and $q_{\widehat{\mathsf{G}_{2k+i}}}(\mathbf{x_i})$ is the canonical query of the digraph $\widehat{\mathsf{G}_{2k+i}}$, $i \in \{0, 1\}$. In what follows we will not usually spell out GAV constraints but rather describe their intended meaning. In all cases it would be easy to write down the GAV constraint by using canonical queries as in the example above.
- A GAV constraint stating that, if for every $1 \leq k < N$, G_{2k+b_k} is homomorphic to the source instance, and the source instance contains tuple $R(u_0, \ldots, u_{r-1})$, then $T(u_0 \ldots u_0)$ holds in the target instance.

Let us define $\mathcal{L}_N$ to be the set containing all $\mathcal{M}_{\mathbf{b}}$, where $\mathbf{b}$ is a N-ary boolean tuple. It is easy to see that every pair of GAV schema mappings in $\mathcal{L}_N$ is logically non-equivalent.

We claim that every example can be labeled in such a way that at most one element in $\mathcal{L}_N$ is inconsistent with the labeling. It follows that, for any learning algorithm that makes less than $2^N - 1$ membership queries, an adversary could pick a target concept in $\mathcal{L}_N$ for which the algorithm would fail. Since the size of the GAV schema mappings in $\mathcal{L}_N$ is bounded by a polynomial in N, it follows that there does not exists any algorithm that learns $\mathrm{GAV}(\mathbf{S}, \mathbf{T})$ using a polynomial number of queries *even if we allow unbounded computational power*.

It remains to prove the claim. Let (I, J) be any example. First, if for every tuple $R(u_0, \ldots, u_{r-1})$ holding in I we have that $T(u_0, \ldots u_0)$ holds in J or for some $0 \leq k < N$ none of $\widehat{\mathsf{G}_{2k}}$ and $\widehat{\mathsf{G}_{2k+1}}$ is homomorphic to I, then we label the example positively. Clearly, in this case, all GAV schema mappings in $\mathcal{L}_N$ are consistent with the labeling. Otherwise, we label the example negatively. If, for some $0 \leq k < N$, $\widehat{\mathsf{G}_{2k}}$ and $\widehat{\mathsf{G}_{2k+1}}$ are homomorphic to I, then, again, all GAV schema mappings in $\mathcal{L}_N$ are consistent with the labeling. We are left in the case in which, for every $0 \leq k < N$, exactly one of $\widehat{\mathsf{G}_{2k}}$ and $\widehat{\mathsf{G}_{2k+1}}$ is homomorphic to I and, furthermore, I contains a tuple $R(u_0, \ldots, u_{r-1})$ such that $T(u_0 \ldots u_0)$ does not hold in J. In this case, there is exactly one GAV schema mapping in $\mathcal{L}_N$ inconsistent with the labeling, namely, $\mathcal{M}_{(b_0, \ldots, b_{N-1})}$ where $b_k \in \{0, 1\}, 0 \leq k < N$ is such that $\mathsf{G}_{2k+b_k} \rightarrow I$. □

4.2 Hardness of efficient learning with equivalence queries

Next, we prove Part 1b of Theorem B in the Introduction.

THEOREM 4.4. *Let* $\mathbf{S}$ *be a source schema and* $\mathbf{T}$ *a target schema. If* $\mathbf{S}$ *contains a relation symbol of arity at least two, then* $\mathrm{GAV}(\mathbf{S}, \mathbf{T})$ *is not efficiently exactly learnable with equivalence queries.*

PROOF. *(Sketch)* We will derive the non-learnability result from an analogous result for Monotone DNF formulas (i.e., formulas of propositional logic that are disjunctions of conjunctions of atoms) proven by Angluin [6]. We will show how, out of a hypothetical efficient exact learning algorithm with equivalence queries for $\mathrm{GAV}(\mathbf{S}, \mathbf{T})$, we can construct an efficient exact learning algorithm with equivalence queries for Monotone DNF formulas.

In the proof, we can assume that the number N of variables of the target monotone DNF formula is known (as it can be obtained from a counterexample of an equivalence query). Let us fix a set $p_1, p_2, \ldots, p_N$ be a set of proposition letters, out of which Monotone DNF formulas are built.

Define G_∞ to be the disjoint union of all digraphs $\mathsf{G}_i \times \mathsf{G}_j$, with $1 \leq i < j \leq N$. It is a well known property of homomorphisms that if H is a digraph such that $H \rightarrow \mathsf{G}_i$ and $H \rightarrow \mathsf{G}_j$ with $1 \leq i < j \leq N$, then $H \rightarrow \mathsf{G}_i \times \mathsf{G}_j$ (and hence $H \rightarrow \mathsf{G}_\infty$). Furthermore, it is also known that if $1 \leq i < j \leq N$, then there are homomorphisms from $\mathsf{G}_i \times \mathsf{G}_j$ to G_i and to G_j; this implies that that $\mathsf{G}_i \not\rightarrow \mathsf{G}_\infty$, for all $1 \leq i \leq N$, since if $\mathsf{G}_i \rightarrow \mathsf{G}_\infty$, it would follow that $\mathsf{G}_i \rightarrow \mathsf{G}_j$ for some $1 \leq j \leq N$ with $j \neq i$, which we know not to be the case.

To each propositional valuation (i.e., a truth assignment from proposition letters $p_1, \ldots, p_N$ to truth values) V, we associate a source instance I_V defined to be a disjoint union of $\widehat{\mathsf{G}_\infty}$ with all $\widehat{\mathsf{G}_k}$ with $k \geq 1$ such that $V(p_k) = 1$. Furthermore, to every labeled valuation (V, ℓ), we associate the labeled $\mathrm{GAV}(\mathbf{S}, \mathbf{T})$ example $((I_V, \emptyset), \neg\ell)$, where $\neg\ell$ is the opposite of ℓ, i.e., $\neg 0 = 1$ and $\neg 1 = 0$.

Similarly, to every Monotone DNF formula ϕ, we associate a corresponding GAV schema mapping $\mathcal{M}_\phi$, namely, the schema mapping that contains for each disjunct $p_{k_1} \wedge \cdots \wedge p_{k_i}$ of ϕ, the GAV constraint stating that, if $I_{k_1}, \ldots, I_{k_i}$ all homomorphically map into the source instance, and the source instance contains a tuple $R(u_0, \ldots, u_{r-1})$, then $T(u_0 \ldots u_0)$ holds in the target instance. Recall that such a GAV constraint can be easily constructed using the canonical conjunctive queries of the instances $I_{k_1}, \ldots, I_{k_i}$.

It is easy to see that the following holds.

FACT 4.5. *A propositional valuation V is a positive example for a Monotone DNF formula ϕ if and only if $(I_V, \emptyset)$ is a negative example for $\mathcal{M}_\phi$.*

In what follows, we will be interested only in data examples (I, J) such that $I = I_V$ for some valuation V and $J = \emptyset$. It turns out that every GAV schema mapping $\mathcal{M}$ can be simplified to a GAV schema mapping $\widehat{\mathcal{M}}$ of a special form, such that $\mathcal{M}$ and $\widehat{\mathcal{M}}$ behave the same on such data examples. Intuitively, what will be special about $\widehat{\mathcal{M}}$ is that the left-hand side of each constraint is isomorphic to (the canonical query of) $\widehat{G}$ for some digraph G. Concretely, for every GAV constraint C of the form $\forall \mathbf{x}(\phi \rightarrow \psi)$, let $\widehat{C}$ be the following GAV constraint: if ϕ contains an atom with a relation other than R, then $\widehat{C}$ is undefined. Otherwise, $\widehat{C}$ is the result of replacing, in every atom (which must necessarily be over predicate R) the last $r-2$ variables by new fresh variables. For every GAV schema mapping $\mathcal{M}$, we let $\widehat{\mathcal{M}}$ be the GAV schema mapping consisting of all GAV constraints $\widehat{C}$ (if defined) with $C \in \mathcal{M}$. It is easy to see that the following holds.

FACT 4.6. *For every valuation V, we have that $(I_V, \emptyset)$ satisfies $\mathcal{M}$ if and only it satisfies $\widehat{\mathcal{M}}$.*

For every GAV schema mapping $\mathcal{M}$, we define $\phi_\mathcal{M}$ to be the monotone DNF formula constructed as follows: For each GAV constraint of $\widehat{\mathcal{M}}$, we take its left-hand side, and split it into connected components. For each connected component, which will be of the form $\widehat{X}$ for some digraph X, we check if (i) X maps homomorphically into G_k for some $1 \leq k \leq N$, and (ii) if X maps homomorphically into G_∞. Note that if (i) is the case for two distinct values $1 \leq k \leq n$, then (ii) holds, as we have observed earlier. Now, if (ii) holds, then we disregard this component $\widehat{X}$. Otherwise, if (i) does not hold for any natural number k, we write down $\bot$. Otherwise, there is a unique value $1 \leq k \leq N$ for which X maps into G_k, and in this case, we write down proposition letter p_k. Doing this for each connected component of the left-hand side of the GAV constraint, we obtain a (possibly empty) set of proposition letters or $\bot$. We take their conjunction. Finally, we do this for each GAV constraint of $\mathcal{M}$, obtaining a disjunction of conjunctions of proposition letters (disjuncts containing $\bot$ are removed), i.e., a Monotone DNF formula.

The following follows easily from the definition of $\phi_\mathcal{M}$ and Fact 4.6.

FACT 4.7. *A valuation V is a positive example for $\phi_\mathcal{M}$ if and only if $(I_V, \emptyset)$ is a negative example for $\mathcal{M}$.*

To avoid cumbersome details, we will not specify how monotone DNFs and GAV schema mappings are represented as strings. For our argument, it is only necessary to know that, if we fix any reasonable representation systems, there exists a polynomial p such that for every monotone DNF ϕ, $\text{size}(\mathcal{M}_\phi) \leq p(\text{size}(\phi))$. Even more strongly, from the fact that the size of G_k is polynomial in k and Lemma 4.2, it follows that:

FACT 4.8. *There are polynomial time algorithms that:*

1. *Compute I_V given a valuation V.*
2. *Compute $\mathcal{M}_\phi$ given a monotone DNF formula ϕ.*
3. *Compute $\phi_\mathcal{M}$ given a GAV schema mapping $\mathcal{M}$.*

Finally, by putting things together, we obtain the desired result: let $\texttt{alg}'$ be a hypothetical efficient exact learning algorithm for $\text{GAV}(\mathbf{S}, \mathbf{T})$ with equivalence queries. We use it to design an efficient exact learning algorithm $\texttt{alg}$ for Monotone DNF formulas with equivalence queries as follows:

Let n be the input of $\texttt{alg}$ and let ϕ_g be the goal monotone DNF. Procedure $\texttt{alg}$ simulates algorithm $\texttt{alg}'$ with input $p(n)$. Each equivalence query request of $\texttt{alg}'$, say with GAV schema mapping $\mathcal{M}$, is processed in the following way: $\mathcal{M}$ is translated to a candidate Monotone DNF formula $\phi_\mathcal{M}$, which is then used to call the equivalence oracle of $\texttt{alg}$. In case of positive answer then $\texttt{alg}$ stops and reports $\phi_\mathcal{M}$. Otherwise, the counterexample, (V, ℓ), provided is translated into a GAV example $((I_V, \emptyset), \neg\ell)$ which is fed back to $\texttt{alg}'$ as the answer of its equivalence query.

It is easy to show the correctness of $\texttt{alg}$. Facts 4.5 and 4.7 guarantee that if (V, ℓ) is a counterexample to the equivalence of $\phi_\mathcal{M}$ and ϕ_g, then $(I_V, \emptyset), \neg\ell)$ is a counterexample of the equivalence of $\mathcal{M}_{\phi_g}$ and $\mathcal{M}$. Consequently, in time polynomial in $p(n)$ (and hence in n), $\texttt{alg}'$ must stop and produce a GAV schema mapping $\mathcal{M}$ equivalent to $\mathcal{M}_{\phi_g}$. Again, from Facts 4.5 and 4.7, it follows that ϕ_g and $\phi_\mathcal{M}$ are logically equivalent. □

It is worth pointing out that the proof in [6] shows something stronger. Indeed, Angluin shows that monotone DNF are not learnable with a polynomial number of equivalence queries even if the learner has unbounded computation power. Notice that this implies that Fact 4.8 is not necessary in our proof (although it will turn out to be necessary in the proof of hardness for efficient PAC-learning that we will give next). Finally, it follows that GAV schema mappings are not learnable with equivalence queries in this stronger sense.

4.3 Hardness of efficient PAC learning

Here, we prove Part 1 of Theorem C in the Introduction.

THEOREM 4.9. *Let $\mathbf{S}$ be a source schema and $\mathbf{T}$ a target schema. If $\mathbf{S}$ contains a relation symbol of arity at least two, then $\text{GAV}(\mathbf{S}, \mathbf{T})$ is not efficiently PAC learnable, provided* RP $\neq$ NP.

PROOF. We use an argument similar to the one used in the proof given in the previous section. Let $\texttt{alg}'$ be a hypothetical efficient PAC-learning algorithm for $\text{GAV}(\mathbf{S}, \mathbf{T})$. We shall prove that the following algorithm, $\texttt{alg}$, correctly PAC-learns monotone DNF formulas; this will contradict the fact that monotone DNF formulas are not efficiently PAC-learnable, if RP $\neq$ NP, as shown in [1].

Let n, ϵ, and δ be the input of $\texttt{alg}$. Procedure $\texttt{alg}$ simulates algorithm $\texttt{alg}'$ with input $p(n)$, ϵ, and δ, answering every example call of $\texttt{alg}'$ by asking its own oracle for an example (V, ℓ) and then answering example $((I_V, \emptyset), \neg\ell)$ to $\texttt{alg}'$. Procedure $\texttt{alg}$ proceeds in this fashion until $\texttt{alg}'$ stops and produces a hypothesis $\mathcal{M}$. When this happens, $\texttt{alg}$ itself stops and reports $\phi_\mathcal{M}$.

Let us show the correctness of the algorithm. Let ϕ_g and D be the unknown goal monotone DNF and unknown probability distribution for $\texttt{alg}$. From the description of $\texttt{alg}$, it follows that $\texttt{alg}'$ is fed with examples drawn according to the probability distribution D' that sets $D'(I, J)$ to $D(V)$ if $I = I_V$ and $J = \emptyset$ and 0 in any other case. Furthermore, by Fact 4.5 these examples are labeled according to $\mathcal{M}_{\phi_g}$, which has size at most $p(n)$. Hence, under this circumstances, $\texttt{alg}'$ should stop in time polynomial in n and, with probability at least $1-\delta$, output a hypothesis $\mathcal{M}$ which has error (with respect to D' and $\mathcal{M}_{\phi_G}$) not larger than ϵ.

From the definition of D' and Fact 4.7 it follows that the error of $\mathcal{M}$ with respect to $\mathcal{M}_{\phi_g}$ and D' is the same than the error of $\phi_\mathcal{M}$ with respect to ϕ_g and D. This completes the proof. □

5. UNARY SOURCE SCHEMAS

In this section, we prove that if the source schema $\mathbf{S}$ consists of unary relations only, then $\mathrm{GAV}(\mathbf{S}, \mathbf{T})$ is efficiently exactly learnable using only membership queries or using only membership queries; $\mathrm{GAV}(\mathbf{S}, \mathbf{T})$ is also efficiently PAC learnable. These results, which were stated in Parts 2a and 2b of Theorem B and in Part 2 of Theorem C in the Introduction, imply that the requirement that the source schema contains a relation of arity at least two is necessary for the negative learnability results in Section 4.

THEOREM 5.1. *Let $\mathbf{S}, \mathbf{T}$ be schemas such that all relations in $\mathbf{S}$ are unary. Then the concept class $\mathrm{GAV}(\mathbf{S}, \mathbf{T})$ of all GAV schema mappings from $\mathbf{S}$ to $\mathbf{T}$ is efficiently exactly learnable with membership queries.*

PROOF. *(Sketch)* If $\mathbf{S}$ and $\mathbf{T}$ are schemas such that all relations in $\mathbf{S}$ are unary, then $\mathrm{GAV}(\mathbf{S}, \mathbf{T})$ can contain only finitely many GAV schema mappings up to logical equivalence. Indeed, let n be the number of unary relations in $\mathbf{S}$, and let k be the maximum arity of the relations in $\mathbf{T}$. It is not hard to see that every GAV constraint over $\mathbf{S}$ and $\mathbf{T}$ is equivalent to one that has at most $2^n + k$ variables: if there are more than 2^n non-exported variables (i.e., variables that do not occur in the right-hand side of the constraint), then two of these variables must satisfy exactly the same unary predicates according to the left-hand side of the constraint, and hence one of these variables could be removed without affecting the semantics of the constraint. Clearly, there are only finitely many pairwise non-equivalent GAV constraints over $\mathbf{S}$ and $\mathbf{T}$ containing at most $2^n + k$ variables.

We pre-compute a finite maximal list of pairwise logically non-equivalent schema mappings together with, for each pair of such schema mappings, a pair (I, J) of instances on which they disagree. Note that all this depends only on the source and target schema, so that the "pre-computation" is not part of the algorithm itself. The algorithm proceeds by asking a membership query for each data example (I, J) constructed during the pre-computation. From the answers, the algorithm can infer which of the schema mappings is logically equivalent to the goal schema mapping. Here, we use the fact that the list of candidate schema mappings is fixed, and hence the candidates can be evaluated on the examples in polynomial time. □

THEOREM 5.2. *Let $\mathbf{S}, \mathbf{T}$ be schemas such that all relations in $\mathbf{S}$ are unary. Then the concept class $\mathrm{GAV}(\mathbf{S}, \mathbf{T})$ of all GAV schema mappings from $\mathbf{S}$ to $\mathbf{T}$ is efficiently exactly learnable with equivalence queries.*

PROOF. *(Sketch)* The proof relies again on the fact that $\mathrm{GAV}(\mathbf{S}, \mathbf{T})$ contains only finitely many GAV schema mappings up to logical equivalence. The algorithm for exactly learning $\mathrm{GAV}(\mathbf{S}, \mathbf{T})$ with equivalence queries works as follows: it maintains a hypothesis $\mathcal{H}$, which is a GAV schema mapping that "overestimates" the goal schema mapping $\mathcal{M}^g$ in the sense that $\mathcal{H}$ logically implies $\mathcal{M}^g$. Initially, $\mathcal{H}$ contains all (finitely many) GAV constraints. The algorithm calls the equivalence oracle to test if the hypothesis is logically equivalent to the goal schema mapping. If this is the case, the algorithm terminates and outputs $\mathcal{H}$. Otherwise, the algorithm receives from the oracle a counterexample (I, J), which must be a positive example for $\mathcal{M}^g$ that is a negative example for $\mathcal{H}$. Since the number of variables in the GAV constraints of $\mathcal{H}$ is bounded, the algorithm can then find in polynomial time all constraints $C \in \mathcal{H}$ that are violated by (I, J). It removes these constraints, and starts again with the updated hypothesis. This process must terminate after a bounded number of steps (where the bound is the number of pairwise non-equivalent GAV constraints over $\mathbf{S}$ and $\mathbf{T}$ containing at most $2^n + k$ variables). □

In the case in which all relations in $\mathbf{S}$ are unary, the concept class $\mathrm{GAV}(\mathbf{S}, \mathbf{T})$ is easily seen to be polynomial-time evaluable. Hence, the following result follows directly from Theorems 2.1 and 5.2.

COROLLARY 5.3. *Let $\mathbf{S}, \mathbf{T}$ be schemas such that all relations in $\mathbf{S}$ are unary. Then the concept class $\mathrm{GAV}(\mathbf{S}, \mathbf{T})$ of all GAV schema mappings from $\mathbf{S}$ to $\mathbf{T}$ is efficiently PAC learnable.*

6. VARIATIONS

6.1 C-acyclic GAV schema mappings

In [40], a subclass of GAV constraints was introduced, called *c-acyclic GAV constraints*, which is more well behaved when it comes to the problem of characterizing schema mappings by data examples. Specifically, it was shown in [40] that a GAV schema mapping $\mathcal{M}$ is uniquely characterizable by a finite set of universal examples if and only if $\mathcal{M}$ is logically equivalent to a schema mapping specified by c-acyclic GAV constraints. This raises the question: does the good behavior of schema mappings specified by c-acyclic GAV constraints extend to efficient learnability? Here, we address this question.

We first introduce some terminology. Let $C = \forall \mathbf{x}(\phi \rightarrow \psi)$ be a GAV constraint. We say that a variable $x \in \mathbf{x}$ is *exported* if it occurs in ψ, and *non-exported* otherwise. The *incidence graph* of ϕ is the bipartite graph whose set of nodes consists of the variables of $\mathbf{x}$ and the atoms in ϕ, and there is an edge between a variable and an atom if the variable occurs in the atom. We say that C is *acyclic* if the incidence graph of ϕ is acyclic and no variable occurs in the same atom twice. We say that the GAV constraint C is *c-acyclic* if every cycle in the incidence graph of ϕ passes though an exported variable, and no non-exported variable occurs in the same atom twice. For example, the GAV constraint $\forall x, y(E(x, y) \wedge E(y, x) \rightarrow F(x, x))$ is c-acyclic, while the GAV constraint $\forall x, y(S(x) \wedge R(y, y) \rightarrow T(x))$ is not. Observe that acyclicity implies c-acyclicity, but not the other way around. A GAV schema mappings is said to be *acyclic*, or *c-acyclic* if all its GAV constraints are. For all schemas $\mathbf{S}, \mathbf{T}$, we define $\mathrm{GAV\text{-}CA}(\mathbf{S}, \mathbf{T})$ to be the concept class that is the restriction of $\mathrm{GAV}(\mathbf{S}, \mathbf{T})$ to sets of c-acyclic GAV constraints.

Note that the concept class $\mathrm{GAV\text{-}CA}(\mathbf{S}, \mathbf{T})$, unlike $\mathrm{GAV}(\mathbf{S}, \mathbf{T})$, is polynomial-time evaluable for all schemas $\mathbf{S}, \mathbf{T}$. This follows from the well known fact that conjunctive queries of bounded tree-width can be evaluated in polynomial time (e.g., see [15, 36]). It can be shown that the tree-width of the left-hand side of a c-acyclic GAV constraint is bounded by the maximal arity of the relations in the target schema plus one.

The results listed in Theorem B in the Introduction remain true when $\mathrm{GAV}(\mathbf{S}, \mathbf{T})$ is replaced by $\mathrm{GAV\text{-}CA}(\mathbf{S}, \mathbf{T})$. For the positive results, this is because it is assumed that the source schema consists of unary relations, and in this case, every GAV schema mapping is trivially c-acyclic (in fact, acyclic). For the negative results, this is because the proofs only involve c-acyclic GAV constraints.

It is interesting to note that the proof of Theorem 4.3 establishes something stronger: it shows that there are c-acyclic GAV schema mappings $\mathcal{M}_1, \ldots, \mathcal{M}_k, \ldots$, such that (a) the size of $\mathcal{M}_k$ is bounded by some polynomial $p(k)$; and (b) if D is a set of positive and negative examples that uniquely characterizes $\mathcal{M}_k$ with respect to the class of c-acyclic GAV schema mappings of size at most $p(k)$, then D must contain at least $2^k - 1$ data examples.

We leave it as an open question whether or not Theorem A holds for $\mathrm{GAV\text{-}CA}(\mathbf{S}, \mathbf{T})$. One may think that Theorem A holds for $\mathrm{GAV\text{-}CA}(\mathbf{S}, \mathbf{T})$ because this class is a subclass of $\mathrm{GAV}(\mathbf{S}, \mathbf{T})$ for which Theorem A holds. However, an exact learning algorithm

for a class need not be an exact learning algorithm for a subclass, since the hypotheses produced by the algorithm need not always be members of the subclass. Observe that, since GAV-CA$(\mathbf{S}, \mathbf{T})$ is polynomially evaluable, if Theorem A holds for GAV-CA$(\mathbf{S}, \mathbf{T})$, then Theorem 2.1 will imply that GAV-CA$(\mathbf{S}, \mathbf{T})$ is efficiently PAC learnable with membership queries.

6.2 Universal examples

A *universal* example of a GAV schema mapping $\mathcal{M}$ is a pair (I, J) such that J is a universal solution for I with respect to $\mathcal{M}$. Intuitively, universal examples contain both positive and negative information. For instance, if $\{Q(a)\}$ is a universal solution for $\{R(a)\}$, this implies that $(\{R(a)\}, \{Q(a)\})$ is a positive example and that $(\{R(a)\}, \emptyset)$ is a negative example. In fact, the information that (I, J) is a universal example of some schema mapping is very powerful as it allows us to deduce for every target instance J' whether (I, J') is a positive or a negative example (specifically, (I, J') is a positive example if and only if $J \to J'$).

PROPOSITION 6.1. *Let $\mathcal{M}$ be a GAV schema mapping, let I be a source instance and J a target instance. Let $J|_{adom(I)}$ be the subinstance of J consisting only of the facts that are over the active domain of I. If J is a universal solution for I with respect to $\mathcal{M}$, then so is $J|_{adom(I)}$.*

This shows that, in learning GAV schema mappings from universal examples, we may disregard facts in target instances that are not over the active domain of the source instance, without affecting the learning problem. Hence, in what follows we will consider only universal examples (I, J) such that $adom(J) \subseteq adom(I)$. Recall that every source instance I has exactly one universal solution J with the property that $adom(J) \subseteq adom(I)$, namely, $\mathsf{can\text{-}sol}(I)$.

We can view GAV schema mappings with universal examples as a concept class, if we are willing to generalize the notion of a concept a bit. Before, we have defined a concept to be a function from examples to $\{0, 1\}$. Now, we will view concepts as functions from examples to some infinite set L of labels. In particular, we will view source instances as examples and target instances as labels, and we will say that a GAV schema mapping $\mathcal{M}$ is *consistent* with a labeled example (I, J) if $J = \mathsf{can\text{-}sol}_{\mathcal{M}}(I)$. We define GAV-U$(\mathbf{S}, \mathbf{T})$ to be the concept class where the example space is the set of source instances, the label set is the set of target instances, and each concept is a GAV schema mapping viewed as a function from source instances to their canonical universal solutions.

All notions we defined before still make sense in this more general setting. Note that, since we are now working with an infinite set of labels, we should assume that labels are represented in some representation system, and count them as having a certain length. However, since we may always assume that the label of a source instance is a target instance over the same active domain, the size of the label cannot be larger than a polynomial in the length of the source instance, and hence this will have no adverse consequences.

By a *labeling oracle* we will mean an oracle that returns the label of a given example, according to the goal concept. This is the natural generalization of membership oracle to this setting of more than two labels. In our setting, the labeling oracle, given a source instance, returns the canonical universal solution according to the goal schema mapping. Note that asking a labeling query for a source instance I provides the same information as asking all membership queries (I, J) with J a target instance over the active domain of I, or, equivalently, asking all (polynomially many) membership queries (I, J) with J a target instance consisting of all possible facts over the active domain of I except for one fact. Also, note that, with respect to equivalence queries, there is no essential difference between the concept classes GAV$(\mathbf{S}, \mathbf{T})$ and GAV-U$(\mathbf{S}, \mathbf{T})$. Hence, all results concerning exact learnability transfer from GAV$(\mathbf{S}, \mathbf{T})$ to GAV-U$(\mathbf{S}, \mathbf{T})$, provided that membership queries are replaced by labeling queries. The results concerning PAC learnability transfer as well, as we will show next.

LEMMA 6.2. *Let $\mathbf{S}, \mathbf{T}$ be schemas. Then the following statements are true.*

1. *If* GAV$(\mathbf{S}, \mathbf{T})$ *is efficiently PAC learnable, then so is* GAV-U$(\mathbf{S}, \mathbf{T})$.
2. *If* GAV$(\mathbf{S}, \mathbf{T})$ *is efficiently PAC learnable with membership queries, then* GAV-U$(\mathbf{S}, \mathbf{T})$ *is efficiently PAC learnable with labeling queries. This also holds if we add an oracle to both sides.*

PROOF. For Part 1, let `alg` be an efficient PAC learning algorithm for GAV$(\mathbf{S}, \mathbf{T})$. We shall construct an efficient PAC learning algorithm $\mathtt{alg}_U$ for GAV-U$(\mathbf{S}, \mathbf{T})$ using `alg`. Let n, ϵ, δ be the input parameters of $\mathtt{alg}_U$ and let $\mathcal{M}^g$ and D be the unknown goal concept and distribution for $\mathtt{alg}_U$.

In a first stage, $\mathtt{alg}_U$ computes a good estimation, m, of the size of the maximum number of tuples for $\mathsf{can\text{-}sol}_{\mathcal{M}^g}(I)$ where I is drawn according to D. This is done by requesting r (to be specified later) examples to its random oracle and setting m to be the maximum number of tuples of any target data instance appearing in them. For every I, let n_I be the number of facts of $\mathsf{can\text{-}sol}_{\mathcal{M}^g}(I)$.

FACT 6.3. *If $r \geq (2/\epsilon)\ln(2/\delta)$ then with probability at least $1 - \delta/2$, $(\Pr_{I \in D} n_I \leq m) \geq 1 - \epsilon/2$.*

PROOF. Let M be the smallest value such that $(\Pr_{I \in D} n_I \leq M) \geq 1 - \epsilon/2$. Hence, if we draw (according to D) r examples $I_1, \ldots, I_r$, the probability that for all $1 \leq i \leq r$, the number of facts of $\mathsf{can\text{-}sol}_{\mathcal{M}^g}(I_i)$ is smaller than M is less that $(1 - \epsilon/2)^r \leq \exp(-\epsilon r/2)$. Standard computations show that this can be made smaller than $\delta/2$ if $r \geq (2/\epsilon)\ln(2/\delta)$. $\square$

Let I be a source instance and let $\mathsf{can\text{-}sol}_{\mathcal{M}^g}(I)$ be its canonical universal solution with respect to $\mathcal{M}^g$. Associated to $(I, \mathsf{can\text{-}sol}_{\mathcal{M}^g}(I))$ there is the collection of all data examples (I, J) where J is either $\mathsf{can\text{-}sol}_{\mathcal{M}^g}(I)$ or is obtained from $\mathsf{can\text{-}sol}_{\mathcal{M}^g}(I)$ by removing one fact.

In a second stage, procedure $\mathtt{alg}_U$ simulates `alg` with parameters $\epsilon/2(m+1), \delta/2$, and n. Every time `alg` requests an example, $\mathtt{alg}_U$ asks its own oracle for a random example $(I, \mathsf{can\text{-}sol}^g_{\mathcal{M}}(I))$ and answers `alg` call with $((I, J), \ell)$ where (I, J) is selected at random and uniformly among the set examples associated to $(I, \mathsf{can\text{-}sol}^g_{\mathcal{M}}(I))$ and the label ℓ is 1 if $J = \mathsf{can\text{-}sol}_{\mathcal{M}^g}(I)$, and 0 otherwise. Notice that the label is defined such that it is 1 if and only if $(I, J) \models \mathcal{M}^g$. Procedure $\mathtt{alg}_U$ proceeds in this fashion until `alg` stops and produces a hypothesis $\mathcal{M}$. When this happens $\mathtt{alg}_U$ itself stops and reports $\mathcal{M}$, as well.

Let us show the correctness of the algorithm. By the design of $\mathtt{alg}_U$ it follows that the examples provided to the oracle of `alg` are labeled according to $\mathcal{M}^g$ and drawn according to the distribution D_U defined as follows: for every data example (I, J), we have that $D_U((I, J)) = D(I)/(n_I + 1)$ if (I, J) is an example associated to $(I, \mathsf{can\text{-}sol}^g_{\mathcal{M}}(I))$, and 0 otherwise.

With probability at most $\delta/2$, the hypothesis $\mathcal{M}$ returned by `alg` satisfies $\mathrm{error}_{\mathcal{M}^g, D_U}(\mathcal{M}) \leq \epsilon/2(m+1)$. For every source example I, if $\mathsf{can\text{-}sol}_{\mathcal{M}}(I) \neq \mathsf{can\text{-}sol}_{\mathcal{M}^g}(I)$, then $\mathcal{M}$ and $\mathcal{M}_g$ disagree in at least one of the $n_I + 1$ examples associated to $(I, \mathsf{can\text{-}sol}\mathcal{M}^g(I))$. It follows that with probability at least

$1 - \delta/2$, the error of $\mathcal{M}$ with respect to D and $\mathcal{M}^g$ over the examples I such that $n_I \leq n$ is at most $1 - \epsilon/2$. By combining this with Fact 6.3, we infer that $\text{error}_{\mathcal{M}^g,D}(\mathcal{M}) \leq \epsilon$ with probability at least $1 - \delta$.

Part 2 follows from the fact that $\texttt{alg}_U$ can answer any membership query (I, J) of alg by testing whether $J \subseteq \textsf{can-sol}_{\mathcal{M}^g}(I)$.

THEOREM 6.4. *Let* $\mathbf{S}, \mathbf{T}$ *be two schemas. If* RP $\neq$ NP*, then* GAV-U$(\mathbf{S}, \mathbf{T})$ *is efficiently PAC learnable if and only if* $\mathbf{S}$ *consists only of unary relations.*

PROOF. *(Sketch)* The positive learnability result follows directly from Corollary 5.3 and Lemma 6.2. The negative learnability result is a straightforward adaptation of the proof of Theorem 4.9. All that is needed is to change the function that translates examples of alg to examples of $\texttt{alg}'$ (since now it must return universal examples). In this case, we map every labeled example (V, ℓ) to (I_V, J_V), where I_V is defined as in Theorem 4.9 and J_V is defined to be $\emptyset$ if $\ell = 0$, and to contain all tuples $T(u_1, \ldots, u_1)$ for every $R(u_1, u_2, \ldots)$ in I_V if $\ell = 1$. We omit the details. □

6.3 Hardness of approximating the smallest fitting GAV schema mapping

In [27], a framework was introduced for deriving a GLAV schema mapping on the basis of a single example (I, J). The framework is based on a cost function that, among other parameters, depends on the size of the schema mapping. The main results in [27] establish hardness of various computational tasks, such as computing an optimal GLAV schema mapping for a given example, or even computing the cost of a given GLAV schema mapping and a given example. The same results are shown to hold for GAV schema mappings.

In the same spirit, we consider here the minimization problem of computing, given a set of (positive/negative or universal) examples, a fitting GAV schema mapping whose size is close to minimal, or, more precisely, a GAV schema mapping whose size is polynomial in the size of the smallest fitting GAV schema mapping. We show that this problem is hard, assuming RP $\neq$ NP. Note that, unlike in [27], we require the schema mapping to *exactly* fit the input examples. On the other hand, we do not require an optimal solution, but a solution that is close to optimal.

We first observe that, whereas every consistent set of *universal examples* has a fitting schema mapping which is basically a translation of the examples [4], in the case of *positive and negative examples*, the size of the smallest fitting schema may not be polynomially bounded in the total size of the examples. Consequently, it is reasonable to require the minimization algorithm to be polynomial in the combined size of its input and of the smallest fitting GAV schema mapping.

Hardness results for this minimization problem can be derived from PAC-hardness. First, we need some preliminaries.

Formally, an *Occam algorithm for* $\mathcal{C}$ *with parameters* $\alpha < 1$ *and* $k \geq 1$ is an algorithm alg that takes as input a collection $(x_1, c(x_1)), \ldots, (x_m, c(x_m))$ of examples labeled according to some unknown concept $c \in \mathcal{C}$ and produces a hypothesis h consistent with the input of size at most $m^\alpha n^k$ where n is the size of c. Furthermore, the running time of alg is required to be bounded by a polynomial in n and the size of the input. Blumer et al [12] proved that one can transform every Occam algorithm alg into an efficient PAC learning algorithm $\texttt{alg}'$. The algorithm $\texttt{alg}'$ asks for m random examples with

$$m = \left(\frac{n^k \ln 2 + \ln(2/\delta)}{\epsilon} \right)^{1/(1-\alpha)}$$

that are fed to the Occam algorithm alg. The hypothesis produced by alg is then returned by $\texttt{alg}'$, yielding the following result.

THEOREM 6.5. *[12] If there is an Occam algorithm for some concept class* $\mathcal{C}$*, then* $\mathcal{C}$ *is efficiently PAC learnable.*

An Occam algorithm can be regarded as an algorithm that approximates the smallest fitting GAV schema mapping. Theorems 4.9 and 6.5 yield the following inapproximability result.

THEOREM 6.6. *Let* $\mathbf{S}, \mathbf{T}$ *be schemas such that* $\mathbf{S}$ *contains a relation of arity at least 2. If* RP $\neq$ NP*, then there is no Occam algorithm for* GAV$(\mathbf{S}, \mathbf{T})$.

In the case of universal examples, we can show something stronger, namely, we can show hardness even when the input consists of a single universal example.

THEOREM 6.7. *Let* $\mathbf{S}, \mathbf{T}$ *be schemas such that* $\mathbf{S}$ *contains at least two relations of arity at least 2. Assume* RP $\neq$ NP*. For every* $k > 0$*, there is no polynomial time algorithm that takes as input a consistent data example* (I, J) *and returns a GAV schema mapping for which* (I, J) *is universal with size at most* n^k*, where* n *is the size of the smallest GAV schema mapping for which* (I, J) *is universal.*

PROOF. *(Sketch)* In the case of universal examples, one cannot use directly Theorem 6.5 as it has been stated only for Boolean concepts, i.e., concepts with labels are restricted to be $\{0, 1\}$. However, it is routine to adapt the proof for the case in which this restriction is lifted. With some extra work, one can show how a finite set of universal examples can be encoded into a single universal example, in a way that allows us to transfer our non-approximability result to the case of a single universal example.

Let $\mathbf{S}$ and $\mathbf{T}$ be schemas, and let S be a relation symbol of arity at least 2, not belonging to either of the two schemas. Consider a finite set of universal examples $e_1 = (I_1, J_1), \ldots, e_n = (I_1, J_1)$ over $\mathbf{S}$ and $\mathbf{T}$. Let $e^* = (I^*, J^*)$ be the universal example over schema $\mathbf{S} \cup \{S\}$ and $\mathbf{T}$, where I^* is the disjoint union of $I_1, \ldots, I_n$ expanded with the equivalence relation S satisfying $S(x_1, \ldots, x_m)$ if and only if $x_1, \ldots x_m$ belong to the domain of the same universal example e_i. Let J^* be the disjoint union of $J_1, \ldots, J_n$.

We claim that approximating the shortest fitting GAV schema mapping for the universal example e^* is as hard as approximating the shortest fitting GAV schema mapping for the universal examples $e_1, \ldots, e_n$, up to a polynomial. This is formally expressed in the next claim.

CLAIM: If there is a GAV schema mapping of length n that is consistent with the universal examples $e_1, \ldots, e_n$, then there is a GAV schema mapping of length $O(n^2)$ that is consistent with the universal example e^*. Conversely, if there is a GAV schema mapping of length n that is consistent with e^*, then there is a GAV schema mapping of length at most n that is consistent with $e_1, \ldots, e_n$.

PROOF OF THE CLAIM: Let $\mathcal{M}$ be a GAV schema mapping over the schemas $\{R\}, \mathbf{T}$ that is consistent with the universal examples $e_1, \ldots, e_n$. Define $\mathcal{M}'$ to be the GAV schema mapping over the schemas $\mathbf{S}, \mathbf{T}$ that contains, for each GAV constraint $C \in \mathcal{M}$ of the form $\forall \mathbf{x}(\phi \rightarrow \psi)$, the GAV constraint $\forall \mathbf{x}(\phi \wedge \bigwedge_{x,x' \in \mathbf{x}} S(x, x') \rightarrow \psi)$. It is not hard to see that $\mathcal{M}$ is consistent with (I^*, J^*) and that the length of $\mathcal{M}'$ is at most quadratic in the length of $\mathcal{M}$.

Conversely, let $\mathcal{M}$ be a GAV schema mapping over the schemas $\mathbf{S}, \mathbf{T}$ that is consistent with the universal example (I^*, J^*). Let $\mathcal{M}'$ be obtained from $\mathcal{M}$ by dropping all atoms involving the relation S from the left-hand side of each GAV constraint. We claim

that $\mathcal{M}'$ is consistent with each universal example $e_i = (I_i, J_i)$. Suppose not, i.e., suppose that $\mathcal{M}'$ contains a GAV constraint $C = \forall \mathbf{x}(\phi \rightarrow \psi)$ that is falsified in e_i under some assignment h. By construction of $\mathcal{M}'$, we have that $\mathcal{M}$ contains a GAV constraint that is identical to C except that ϕ contains possibly some number of atoms involving the relation S. But then, it is not hard to see that C' is falsified in (I^*, J^*) under the assignment h (note that h maps all variables of $\mathbf{x}$ into a single "equivalence class" of S). Therefore, we have reached a contradiction. □

Consequently, the problem of approximating the minimum fitting GAV schema mapping within a polynomial is hard, even when restricted to consistent inputs. We point out that these results not only constitute strong inapproximability results, but are also independent of the choice of encoding for schema mappings.

It is also worth noting that, if we allow inconsistent inputs, then we can derive hardness of approximability, although not up to a polynomial factor, directly from the fact that the decision problem (deciding whether there is a fitting schema mapping) is hard [4].

7. CONCLUDING REMARKS

In this paper, we established a new connection between database theory and computational learning theory by first casting the problem of obtaining algorithmically a GAV schema mapping from data examples as a learning problem and then establishing both positive and negative results. With some work, our negative results concerning the learnability of GAV schema mappings extend to similar negative results for the class of all GLAV (Global-and-Local-As-View) schema mappings. It remains to be determined what is the learnability status of LAV (Local-As-View) schema mappings in the various learning models. Also, we leave it as an open question whether GAV schema mappings are efficiently PAC learnable with membership queries (and without an NP oracle). It also remains to be investigated which of our results generalize to the case where the source schema and the target schema are treated as part of the input of the learning problems.

8. REFERENCES

[1] M. Alekhnovich, M. Braverman, V. Feldman, A. R. Klivans, and T. Pitassi. The complexity of properly learning simple concept classes. *J. Comput. Syst. Sci.*, 74(1):16–34, 2008.

[2] B. Alexe, L. Chiticariu, R. J. Miller, and W. C. Tan. Muse: Mapping Understanding and deSign by Example. In *ICDE*, pages 10–19, 2008.

[3] B. Alexe, P. G. Kolaitis, and W. C. Tan. Characterizing schema mappings via data examples. In *PODS*, pages 261–272, 2010.

[4] B. Alexe, B. ten Cate, P. G. Kolaitis, and W. C. Tan. Designing and refining schema mappings via data examples. In *SIGMOD*, pages 133–144, 2011.

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

[6] D. Angluin. Negative results for equivalence queries. *Machine Learning*, 5:121–150, 1990.

[7] D. Angluin, M. Frazier, and L. Pitt. Learning conjunctions of horn clauses. *Mach. Learn.*, 9:147–164, July 1992.

[8] D. Angluin and M. Kharitonov. When won't membership queries help? *J. Comput. Syst. Sci.*, 50(2):336–355, 1995.

[9] M. Anthony and N. Biggs. *An Introduction to Computational Learning Theory*. Cambridge University Press, 1992.

[10] P. Barceló. Logical foundations of relational data exchange. *SIGMOD Record*, 38(1):49–58, 2009.

[11] P. A. Bernstein, T. J. Green, S. Melnik, and A. Nash. Implementing Mapping Composition. *VLDB Journal*, 17(2):333–353, 2008.

[12] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Occam's Razor. *Inf. Process. Lett.*, 24(6):377–380, 1987.

[13] A. Bonifati, E. Q. Chang, T. Ho, V. S. Lakshmanan, and R. Pottinger. HePToX: Marrying XML and Heterogeneity in Your P2P Databases. In *VLDB*, pages 1267–1270, 2005.

[14] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *STOC*, pages 77–90, 1977.

[15] C. Chekuri and A. Rajaraman. Conjunctive query containment revisited. *Theor. Comput. Sci.*, 239:211–229, May 2000.

[16] W. W. Cohen. PAC-learning non-recursive prolog clauses. *Artif. Intell.*, 79(1):1–38, 1995.

[17] W. W. Cohen. PAC-learning recursive logic programs: Efficient algorithms. *J. Artif. Intell. Res. (JAIR)*, 2:501–539, 1995.

[18] W. W. Cohen. PAC-learning recursive logic programs: Negative results. *J. Artif. Intell. Res. (JAIR)*, 2:541–573, 1995.

[19] A. Das Sarma, A. G. Parameswaran, H. Garcia-Molina, and J. Widom. Synthesizing view definitions from data. In *ICDT*, pages 89–103, 2010.

[20] A. Doan. *Learning to Map between Structured Representations of Data*. PhD thesis, Computer Science and Engineering, University of Washington, University of Pennsylvania, 2002.

[21] A. Doan, J. Madhavan, P. Domingos, and A. Y. Halevy. Learning to map between ontologies on the semantic web. In *WWW*, pages 662–673, 2002.

[22] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data Exchange: Semantics and Query Answering. *Theoretical Computer Science (TCS)*, 336(1):89–124, 2005.

[23] W. Fan, F. Geerts, L. V. S. Lakshmanan, and M. Xiong. Discovering conditional functional dependencies. In *ICDE*, pages 1231–1234, 2009.

[24] V. Feldman. Hardness of approximate two-level logic minimization and PAC learning with membership queries. *J. Comput. Syst. Sci.*, 75(1):13–26, 2009.

[25] P. A. Flach and I. Savnik. Database dependency discovery: A machine learning approach. *AI Commun.*, 12(3):139–160, 1999.

[26] G. H. L. Fletcher, M. Gyssens, J. Paredaens, and D. V. Gucht. On the expressive power of the relational algebra on finite sets of relation pairs. *IEEE Trans. Knowl. Data Eng.*, 21(6):939–942, 2009.

[27] G. Gottlob and P. Senellart. Schema mapping discovery from data instances. *J. ACM*, 57(2), 2010.

[28] W. Gutjahr, E. Welzl, and G. Woeginger. Polynomial graph-colorings. *Discrete Appl. Math.*, 35:29–46, 1992.

[29] L. M. Haas, M. A. Hernández, H. Ho, L. Popa, and M. Roth. Clio Grows Up: From Research Prototype to Industrial Tool. In *SIGMOD*, pages 805–810, 2005.

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

[31] L. Hellerstein, K. Pillaipakkamnatt, V. V. Raghavan, and D. Wilkins. How many queries are needed to learn? *J. ACM*, 43(5):840–862, 1996.

[32] K. Hirata. On the hardness of learning acyclic conjunctive queries. In *ALT*, pages 238–251, 2000.

[33] M. J. Kearns and L. G. Valiant. Cryptographic limitations on learning boolean formulae and finite automata. *J. ACM*, 41(1):67–95, 1994.

[34] M. J. Kearns and U. V. Vazirani. *An Introduction to Computational Learning Theory*. The MIT Press, 1997.

[35] P. G. Kolaitis. Schema Mappings, Data Exchange, and Metadata Management. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 61–75, 2005.

[36] P. G. Kolaitis and M. Y. Vardi. Conjunctive-query containment and constraint satisfaction. *J. Comput. Syst. Sci.*, pages 302–332, 2000.

[37] M. Lenzerini. Data Integration: A Theoretical Perspective. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 233–246, 2002.

[38] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB J.*, 10(4):334–350, 2001.

[39] R. E. Schapire. The strength of weak learnability. *Machine Learning*, 5:197–227, 1990.

[40] B. ten Cate, P. G. Kolaitis, and W. C. Tan. Database constraints and homomorphism dualities. In *CP*, 2010.

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

[42] L. G. Valiant. Learning disjunctions of conjunctions. In *IJCAI*, pages 560–566, 1985.

[43] L. Yan, R. J. Miller, L. M. Haas, and R. Fagin. Data-Driven Understanding and Refinement of Schema Mappings. In *SIGMOD*, pages 485–496, 2001.