The Exact Complexity of the First-Order Logic Definability Problem

MARCELO ARENAS, Pontificia Universidad Católica de Chile
GONZALO I. DIAZ, University of Oxford

We study the *definability problem* for first-order logic, denoted by FO-Def. The input of FO-Def is a relational database instance I and a relation R; the question to answer is whether there exists a first-order query Q (or, equivalently, a relational algebra expression Q) such that Q evaluated on I gives R as an answer.

Although the study of FO-Def dates back to 1978, when the decidability of this problem was shown, the exact complexity of FO-Def remains as a fundamental open problem. In this article, we provide a polynomial-time algorithm for solving FO-Def that uses calls to a graph-isomorphism subroutine (or oracle). As a consequence, the first-order definability problem is found to be complete for the class **GI** of all problems that are polynomial-time Turing reducible to the graph isomorphism problem, thus closing the open question about the exact complexity of this problem. The technique used is also applied to a generalized version of the problem that accepts a finite set of relation pairs, and whose exact complexity was also open; this version is also found to be **GI**-complete.

CCS Concepts: • **Information systems → Relational database query languages**; • **Theory of computation → Problems, reductions, and completeness;**

Additional Key Words and Phrases: Definability problem, expressiveness, relational algebra, first-order logic

ACM Reference Format:
Marcelo Arenas and Gonzalo I. Diaz. 2016. The exact complexity of the first-order logic definability problem. ACM Trans. Database Syst. 41, 2, Article 13 (May 2016), 14 pages.
DOI: http://dx.doi.org/10.1145/2886095

1. INTRODUCTION

In a typical relational database querying scenario, a database instance I and a query Q are given and the objective is to *answer the query*, that is, calculate the answer relation R that Q produces for I. There are many real-world situations, though, in which the database I and answer relation R are known and it is the query that is unknown. For example, it is common for the results of a query to be published without the precise query being made available, requiring *reverse engineering* [Tran et al. 2009; Zhang et al. 2013]. In other cases, users of a database system may have difficulties formulating a query, in which case it is desirable to have the ability to *infer* or *learn* the query by having the user specify tuples that they want included in the result (i.e., positive examples), tuples that they want excluded from the result (i.e., negative examples), or a combination of both. Query learning has been studied in relational databases [Abouzied et al. 2013; Bonifati et al. 2014a] as well as in other data models such as

M. Arenas was funded by the Millennium Nucleus Center for Semantic Web Research under Grant NC120004, and G. I. Diaz by Becas Chile of CONICYT Chile.
Authors' addresses: M. Arenas, Departamento de Ciencia de la Computación, Pontificia Universidad Católica de Chile, Vicuna Mackenna 4860, Edificio San Agustín, 4to piso, Macul 7820436, Santiago, Chile; email: marenas@ing.puc.cl; G. I. Diaz, Department of Computer Science, University of Oxford, Wolfson Building, Parks Road, Oxford OX1 3QD, United Kingdom; email: gonzalo.diaz@cs.ox.ac.uk.

DOI: http://dx.doi.org/10.1145/2886095

XML [Staworko and Wieczorek 2012; Cohen and Weiss 2013; Staworko and Wieczorek 2015], graph databases [Bonifati et al. 2015], and big data [Bonifati et al. 2014b]. This situation also arises in data-integration scenarios, in which example source and target instances are used to derive data-mapping queries between source and target [Bilke and Naumann 2005; Gottlob and Senellart 2010; Qian et al. 2012; ten Cate et al. 2013]. Finally, a user may need to check whether a relation R is *redundant* in a database instance I, in the sense that there exists a query Q that produces R when evaluated over the other relations in I, implying that the information in R can be deduced from the other relations in the instance I [Ferrarotti et al. 2009].

A common ground for the different query-discovery scenarios presented is the *definability problem* for a query language $\mathcal{Q}$. This decision problem takes as input an appropriate database instance I and answer R, and asks whether there exists a query $Q \in \mathcal{Q}$ such that Q evaluated on I results in R. Here, the semantics of the query language $\mathcal{Q}$ determines what appropriate database instances and answer sets are. In the case of first-order logic (without constants and without a linear order on the domain) we denote the definability problem FO-Def; the input is a relational database instance I and an answer relation R. The definability problem has been studied for relational databases and first-order logic (or, equivalently, relational algebra), as well as for other data models and query languages. In particular, this problem has been studied for nested relational databases and nested relational algebra [Gucht 1987; Gyssens et al. 1989], for XML and XPath [Gyssens et al. 2006; Fletcher et al. 2015], and for graph databases and conjunctive regular path queries [Antonopoulos et al. 2013].

The study of the computational complexity of FO-Def dates back to 1978 [Paredaens 1978; Bancilhon 1978], when a *semantic characterization* of the problem, based on automorphisms, placed FO-Def in **coNP** (see Section 3 and Van den Bussche [2001] for more details). Although this provided a complexity upper bound for the problem, an exact complexity result has not been found since then [Fletcher et al. 2009; ten Cate and Dalmau 2015]. In particular, the problem was never found to be **coNP**-hard. Despite the open question for the first-order logic case, the corresponding definability problem for conjunctive queries was determined to be **coNEXPTIME**-complete [Willard 2010]. Here, the complexity upper bound (i.e., the inclusion in **coNEXPTIME**) stems from an analogous semantic characterization of the CQ definability problem in terms of polymorphisms [Jeavons et al. 1999]. In a different direction, a natural generalization of FO-Def, dubbed BP-Pairs [Fletcher et al. 2009], accepts a finite set of relation pairs $\{(S_1, R_1), \ldots, (S_n, R_n)\}$ and asks whether there exists a first-order query Q such that $Q(S_i) = R_i$ for all $i \in [1, n]$. By means of an analogous semantic characterization, the authors found BP-Pairs to be included in **coNP**.

In this article, we provide a novel polynomial-time algorithm for the first-order logic definability problem, which uses calls to a graph-isomorphism subroutine (oracle). As a consequence of the existence of this algorithm, FO-Def is found to be included in the complexity class **GI** (defined as the set of all languages that are polynomial-time Turing reducible to the graph isomorphism problem). Moreover, we also show that FO-Def is **GI**-hard, which allows us to conclude that FO-Def is **GI**-complete. This allows us to close the open question regarding the exact complexity of FO-Def. This result also has consequences in practical applications, as implementations for the definability problem may now take advantage of algorithmic optimizations for the graph isomorphism problem [Piperno 2008; McKay and Piperno 2014; Babai 2015]. For example, for several restricted classes of graphs, it is possible to solve the isomorphism problem in polynomial time [Grohe 2012]; this performance will be inherited by definability problem algorithms that use the characterization presented in this article. Finally, we find that the technique used is also applicable to the BP-Pairs problem and show that it is also included in **GI**, solving a problem left open in Fletcher et al. [2009].

2. PRELIMINARIES

Let U be an infinite countable universe. A *relational schema* $\mathbf{R} = \{R_1, \ldots, R_m\}$ is a set of *relation names*, each with an associated *arity*, denoted by $\text{arity}(R_i)$. Given a relational schema, a *relational instance* I *over* $\mathbf{R}$ is a set of *relations* $\{R_1^I, \ldots, R_m^I\}$, with each R_i^I a finite subset of $\mathsf{U}^{\text{arity}(R_i)}$. The *active domain of* I, denoted by $\mathbf{adom}(I)$, is the set of elements of U that appear in some relation of I (we define the active domain of a relation analogously).

We assume familiarity with the syntax and semantics of first-order logic [Abiteboul et al. 1995; Enderton 1972]. Let $\mathbf{R}$ be a relational schema and I an instance over $\mathbf{R}$. A k-ary FO-query Q over $\mathbf{R}$ is given by an FO-formula $\varphi(\bar{x})$, where $\bar{x} = (x_1, \ldots, x_k)$ is the tuple of free variables of φ. Moreover, the evaluation of Q over I, denoted by $Q(I)$, is defined as the set of tuples $\bar{a}$ such that $\varphi(\bar{a})$ holds in I.

Example 2.1. Consider the relational schema $\mathbf{R} = \{\text{Person}, \text{Knows}\}$ with arities 1 and 2, respectively. Also, consider the relational instance $I = \{\text{Person}^I, \text{Knows}^I\}$ defined as follows:

Person^I
Ada
John
Dana
Peter

Knows^I	
Ada	John
John	Ada
Dana	Peter

Then, a query Q_1 given by FO-formula $\text{Person}(x)$ returns the list of persons in I, while a query Q_2 given by FO-formula $\exists y\, (\text{Person}(x) \wedge \text{Knows}(x, y) \wedge x \neq y)$ returns the list of persons in I that know someone else.

Given instances I_1, I_2 over a relational schema $\mathbf{R}$, a function $f : \mathsf{U} \to \mathsf{U}$ is an *isomorphism from* I_1 *to* I_2 if and only if (i) f is a bijection and (ii) for every $R \in \mathbf{R}$ such that $\text{arity}(R) = n$, and for every $t \in \mathsf{U}^n$, it is the case that $t \in R^{I_1}$ if and only if $f(t) \in R^{I_2}$, where $f(t)$ is defined as $(f(a_1), \ldots, f(a_n))$ if $t = (a_1, \ldots, a_n)$. Given an instance I over a relational schema $\mathbf{R}$, a function $f : \mathsf{U} \to \mathsf{U}$ is an *automorphism of* I if f is an isomorphism from I to I. The notions of isomorphism and automorphism for a relation are defined analogously. In what follows, we use $\textsc{Aut}(I)$, $\textsc{Aut}(R)$ to denote the set of automorphisms for an instance I and a relation R, respectively.

Given a tuple $t = (a_1, \ldots, a_n)$, define the i^{th} *prefix* of t in the following way:

$$\pi_{\leq i}(t) = \begin{cases} (a_1, \ldots, a_i) & \text{if } 1 \leq i \leq n, \\ () & \text{otherwise.} \end{cases}$$

In order to extract only one column, we use the notation $\pi_i(t) = a_i$. We also allow an overloaded version of the operator, where R is a relation of arity n:

$$\pi_{\leq i}(R) = \{\pi_{\leq i}(t) \mid t \in R\}.$$

In other words, $\pi_{\leq i}(R)$ is the image of every tuple $t \in R$ under $\pi_{\leq i}$.

The Graph Isomorphism Problem and the Complexity Class **GI**. The graph isomorphism problem is defined as $\textsc{Graph-Iso} = \{(G_1, G_2) \mid G_1 \text{ and } G_2 \text{ are isomorphic graphs}\}$ [Arora and Barak 2009; Köbler et al. 1993]. A major open problem in computational complexity is to determine the exact complexity of this problem, in particular, whether it can be solved in polynomial time, it is **NP**-complete, or it is **NP**-intermediate [Köbler et al. 1993].

The problem $\textsc{Graph-Iso}$ gives rise to the complexity class $\mathbf{GI} = \{L \mid L \leq_T^p \textsc{Graph-Iso}\}$, where $L_1 \leq_T^p L_2$ indicates that there is a polynomial-time Turing reduction from the decision problem L_1 to the decision problem L_2. In other words, **GI** is the class of all problems L that can be solved in polynomial time by an algorithm that uses an oracle (or subroutine) for the graph isomorphism problem [Aaronson et al. 2005]. Furthermore, a decision problem L is said to be **GI**-hard if and only if $L' \leq_T^p L$ for every problem $L' \in$ **GI**. Note that this is a relaxation with respect to the traditional definition of hardness for **NP**, as only a Turing reduction is required (as opposed to a polynomial-time many-to-one reduction for the standard notion of hardness). Examples of **GI**-complete problems that will be used in this article are the relational instance isomorphism problem, $\text{REL-ISO} = \{(I_1, I_2) \mid I_1$ and I_2 are isomorphic relational instances$\}$ [Zemlyachenko et al. 1985], and the automorphism with one antifixed point problem, $\text{AUT-1-AFP} = \{(G, v) \mid G = (V, E)$ is a graph with $v \in V$ such that there exists an automorphism f of G for which $f(v) \neq v\}$ [Lubiw 1981]. It is important to note that, although $\text{GRAPH-ISO} \in$ **NP**, the class **GI** is not known to be a subset of **NP**, since **GI** is defined in terms of polynomial-time Turing reductions and **NP** is not known to be closed under such reductions (**NP** is known to be closed under polynomial-time many-to-one reductions).

Finally, given a decision problem L, we denote the *complement* of L as $\overline{L}$.

3. THE DEFINABILITY PROBLEM FOR FIRST-ORDER LOGIC

The first-order logic definability problem is defined as $\text{FO-DEF} = \{(I, R) \mid I$ is a relational instance, R is a relation, and there is a first-order query Q such that $Q(I) = R\}$. Note that both the schema $\mathbf{R}$ of I and the arity n of R are not fixed but can be deduced from I and R, respectively, and that the query does not mention any constants. This problem was studied in Paredaens [1978] and Bancilhon [1978], where it was determined that given a relational instance I and a relation R, $(I, R) \in \text{FO-DEF}$ (that is, R is definable from I by a first-order query) if and only if (i) $\textbf{adom}(R) \subseteq \textbf{adom}(I)$ and (ii) $\text{AUT}(I) \subseteq \text{AUT}(R)$. The intuition behind this semantic characterization is as follows. Assume that $Q(I) = R$, where Q is an FO-query. Then, R cannot mention a value that does not occur in I, as first-order logic cannot invent new values. Thus, it should be the case that $\textbf{adom}(R) \subseteq \textbf{adom}(I)$. Moreover, assume that a and b are values occurring in I and h is an automorphism of I such that $h(a) = b$. We know that if we replace in I every value c by $h(c)$, then we obtain the same instance I. Hence, a and b are indistinguishable in I. In particular, these two values cannot be differentiated by Q, as Q is defined by an FO-formula whose vocabulary is $\mathbf{R}$ and, thus, by an FO-formula that does not mention any constant. Hence, given that $R = Q(I)$, if any of a or b occurs in R, then the other value has to occur in R, and, more generally, a and b have to be indistinguishable in R. Formally, h has to be an automorphism of; therefore, every automorphism of I has to be an automorphism of R (i.e., $\text{AUT}(I) \subseteq \text{AUT}(R)$).

Example 3.1. Let $\mathbf{R}$ and I be the relational schema and instance shown in Example 2.1, respectively, and assume that R and S are the following relations:

R
John

S	
Ada	John
John	Ada

In this case, the pair $(I, S) \in \text{FO-DEF}$, that is, it is possible to find a first-order query Q such that $Q(I) = S$. In fact, in this case, Q is given by FO-formula $(\text{Knows}(x, y) \wedge \text{Knows}(y, x))$. On the other hand, the relation R is not definable from I by a first-order query. To see why this is the case, note that Ada and John are interchangeable in I, so that if R can be obtained as the result of evaluating an FO-query over I, then Ada has to occur in R as John occurs in R. We can formalize this intuition and prove that $(I, R) \notin \text{FO-DEF}$ by using the semantic characterization of the definability problem in terms of automorphisms. More precisely, consider the function $h : \mathsf{U} \to \mathsf{U}$ such that $h(\text{Ada}) = \text{John}$, $h(\text{John}) = \text{Ada}$, and for any other element $u \in \mathsf{U}$, $h(u) = u$. The function

h thus defined is an automorphism of I. However, h is not an automorphism of R, as $h(\text{John}) \notin R$, from which we conclude that $(I, R) \notin$ FO-DEF.

4. THE EXACT COMPLEXITY OF FO-DEF

From the characterization of FO-DEF in the previous section, it is clear that FO-DEF $\in$ **coNP**, as an automorphism of I that is not an automorphism of R, provides a (polynomially sized) witness to the fact that $(I, R) \notin$ FO-DEF. Although this provides an upper bound for the complexity of this problem, the exact complexity of FO-DEF is an open problem; in particular, the problem is not known to be **coNP**-hard. The following is the main result of this article, in which we close this problem:

THEOREM 4.1. FO-DEF *is* **GI**-*complete.*

This result allows us to revisit the status of the FO-DEF problem with respect to **P** and **NP**. As a first corollary of Theorem 4.1, we can now say that if GRAPH-ISO $\in$ **P**, then it will also be the case that FO-DEF $\in$ **P**. As second corollary of Theorem 4.1, we obtain strong evidence against the **coNP**-hardness of FO-DEF. Recall that if $\mathcal{C}$ is a complexity class, then $\mathbf{NP}^{\mathcal{C}}$ is the class of decision problems that can be solved in polynomial time by a nondeterministic Turing machine with an oracle for a decision problem $L \in \mathcal{C}$. Moreover, recall that the second level of the polynomial hierarchy [Stockmeyer 1976] consists of the complexity classes $\Sigma_2^p = \mathbf{NP}^{\mathbf{NP}}$ and $\Pi_2^p = \mathbf{co}\Sigma_2^p = \{\overline{L} \mid L \in \Sigma_2^p\}$, which are widely believed to be different. From Theorem 4.1, we have the following.

COROLLARY 4.2. *If* FO-DEF *is* **coNP**-*complete, then* $\Sigma_2^p = \Pi_2^p$.

To understand this corollary, we need to consider the second level of the *low hierarchy* of **NP** [Schöning 1983; Hemaspaandra 1993]. Let $\mathbf{Low}_2$ be the class of decision languages $L \in \mathbf{NP}$ such that

$$\mathbf{NP}^{(\mathbf{NP}^{L})} = \mathbf{NP}^{\mathbf{NP}},$$

that is, the class of languages $L \in \mathbf{NP}$ such that the computational power of the second level of the polynomial hierarchy is not augmented if L is available as an oracle. It is known that GRAPH-ISO $\in \mathbf{Low}_2$ [Schöning 1988], and that if a language in $\mathbf{Low}_2$ is **NP**-complete (under the usual notion of polynomial-time many-to-one reduction), then $\Sigma_2^p = \Pi_2^p$ [Schöning 1983]. From Theorem 4.1 and the fact that GRAPH-ISO $\in$ **NP**, we have that $\overline{\text{FO-DEF}} \in \mathbf{NP} \cap \mathbf{GI}$, from which we conclude that $\overline{\text{FO-DEF}} \in \mathbf{Low}_2$. Hence, if $\overline{\text{FO-DEF}}$ is **NP**-complete, then $\Sigma_2^p = \Pi_2^p$, from which Corollary 4.2 follows.

As a final comment, it is important to mention that Theorem 4.1 holds for any relational query language that is BP-complete [Chandra and Harel 1980; Van Gucht 2009]. Thus, for example, the definability problem for Datalog is also **GI**-complete, for which the input of this problem is an instance I and a relation R, and the question to answer is whether there exists a Datalog program that evaluated over I produces R.

In the rest of this section, we concentrate on proving Theorem 4.1.

4.1. Proof of Theorem 4.1

The **GI**-hardness of FO-DEF is shown via a polynomial-time Turing reduction from AUT-1-AFP; the inclusion of FO-DEF in **GI** is shown via a polynomial-time Turing reduction to REL-ISO. Recall that these problems were defined in Section 2, and that they are both known to be **GI**-complete [Zemlyachenko et al. 1985; Lubiw 1981].

In order to motivate the proof of the inclusion of FO-DEF in **GI**, consider the following algorithm for FO-DEF using an oracle for REL-ISO, which constitutes a failed attempt to show that FO-DEF $\leq_T^p$ REL-ISO. On input (I, R), with $\text{arity}(R) = n$, we wish to show the existence of a function $f \in \text{AUT}(I)$ for which there is a tuple $t \in R$ such that

$f(t) = s$ and $s \notin R$ (note that $s \in \mathbf{adom}(I)^n$).[1] This scenario can be restated in the following way: does there exist a tuple $t \in R$, a *bad* tuple $s \in \mathbf{adom}(I)^n \setminus R$, and an automorphism $f \in \text{AUT}(I)$ such that $f(t) = s$? Then, for every $t = (a_1, \ldots, a_n) \in R$ and $s = (b_1, \ldots, b_n) \in \mathbf{adom}(I)^n \setminus R$, the procedure builds the relational instances I_1 and I_2 from I by *marking*, for every $i \in [1, n]$, the pair a_i, b_i in such a way that any isomorphism from I_1 to I_2 must map a_i to b_i (these markings can be achieved by placing a_i and b_i in fresh unary relations). We then consult the REL-ISO oracle with input (I_1, I_2) to decide whether such an isomorphism exists.

Example 4.3. Consider the pair (I, R) from Examples 2.1 and 3.1. In order to decide whether (I, R) is definable, we iterate over every tuple in , the only such tuple being $t =$ (John). For this fixed t, we iterate over all possible bad tuples $s \in \mathbf{adom}(I) \setminus R$. Upon reaching the case $s =$ (Ada), we build the following instances:

—We first create a fresh relation name Fresh such that arity(Fresh) = arity(R) = 1.
—We prepare an instance I_1 over the relational schema {Person, Knows, Fresh} such that $\text{Person}^{I_1} = \text{Person}^{I}$, $\text{Knows}^{I_1} = \text{Knows}^{I}$, and $\text{Fresh}^{I_1} = \{(\text{John})\}$.
—We prepare an instance I_2 over the relational schema {Person, Knows, Fresh} such that $\text{Person}^{I_2} = \text{Person}^{I}$, $\text{Knows}^{I_2} = \text{Knows}^{I}$, and $\text{Fresh}^{I_2} = \{(\text{Ada})\}$.

We now call a REL-ISO oracle with input (I_1, I_2) in order to decide whether there is an isomorphism from I_1 to I_2. Note that, with the addition of the Fresh^{I} relation, we are actually asking whether there is an automorphism of I that maps John to Ada. The oracle will respond **true**, which will serve as a witness to the nondefinability of (I, R), whereby we return **false**.

The previous algorithm does not constitute a proof that FO-DEF $\leq_T^p$ REL-ISO due to the fact that there are exponentially many bad tuples $s \in \mathbf{adom}(I)^n \setminus R$ to be checked. This problem can be avoided by considering an *incremental* characterization of the first-order definability problem, which we turn to now.

LEMMA 4.4. *Let I be an instance of a relational schema $\mathbf{R}$ and R a relation of arity n such that $\mathbf{adom}(R) \subseteq \mathbf{adom}(I)$. Given $f \in \text{AUT}(I)$, f is not an automorphism of R if and only if there exists a tuple $t \in R$ and an integer $i \in [0, n-1]$ such that*

(1) $f(\pi_{\leq i}(t)) \in \pi_{\leq i}(R)$,
(2) $f(\pi_{\leq i+1}(t)) \notin \pi_{\leq i+1}(R)$.

Intuitively, f is not an automorphism of R if there is a tuple $t \in R$ for which $f(t) \notin R$; however, we can refine this notion by finding the *column* i such that f maps t correctly in the i^{th} prefix (condition (1)), but fails to map t correctly for the $(i+1)^{\text{th}}$ prefix (condition (2)).

PROOF OF LEMMA 4.4. Let I be an instance of a relational schema $\mathbf{R}$ and R a relation of arity n such that $\mathbf{adom}(R) \subseteq \mathbf{adom}(I)$. The following is a well-known characterization of the notion of automorphism of a relation.

CLAIM 1. $f \in \text{AUT}(I)$ *is not an automorphism of R if and only if there exists a tuple $t \in R$ such that $f(t) \notin R$.*

To prove the direction ($\Rightarrow$) of the lemma, we assume that f is not an automorphism of R. In that case, we know by Claim 1 that there is a tuple $t_0 \in R$ such that $f(t_0) \neq t$ for

[1]This would actually shows that (I, R) is *not definable*, but as this is a deterministic algorithm, we may simply return the opposite answer.

every $t \in R$. Then, for every $t \in R$, define k_t as the minimum element of the set

$$\{i \in [1, n] \mid f(\pi_i(t_0)) \neq \pi_i(t)\}.$$

That is, k_t represents the leftmost column for which f fails to map t_0 to t. With the previous, define

$$i_0 = \left(\max_{t \in R} k_t\right) - 1.$$

Then, we have that i_0 satisfies the conditions stated in the lemma:

(1) Let $t' = \text{argmax}_{t \in R}\, k_t$. Then, by definition of i_0 and t', we have that $f(\pi_{\leq i_0}(t_0)) = \pi_{\leq i_0}(t')$ (whereby $f(\pi_{\leq i_0}(t_0)) \in \pi_{\leq i_0}(R)$).
(2) Let $t'' \in R$. Then, by definition of i_0, we have that $f(\pi_{\leq i_0+1}(t_0)) \neq \pi_{\leq i_0+1}(t'')$. As t'' is arbitrary, this implies that $f(\pi_{\leq i_0+1}(t)) \notin \pi_{\leq i_0+1}(R)$.

For the direction ($\Leftarrow$), assume that there exists tuple $t \in R$ and integer $i \in [0, n-1]$ such that items (1) and (2) hold. In particular, item (2) implies that $f(t) \notin R$, whereby f is not an automorphism of R by Claim 1.

We finally have all the necessary ingredients to prove Theorem 4.1.

Proof of Theorem 4.1 We first show that FO-Def $\in$ **GI** by determining that FO-Def $\leq_T^p$ Graph-Iso. We use the result of Lemma 4.4 to produce Algorithm 1, a deterministic polynomial-time algorithm that uses an oracle for the Rel-Iso decision problem.

Let I be a relational instance and R a relation such that arity$(R) = n$, and assume that $\mathbf{adom}(R) \subseteq \mathbf{adom}(I)$. On input (I, R), Algorithm 1 proceeds in a similar way as the naïve algorithm described at the beginning of this section, but trying to show the existence of a function $f \in \text{Aut}(I)$ that fails as an automorphism of R in a specific column of R. More precisely, Algorithm 1 starts by picking the values of i and t in its first two loops, which will be used as stated in Lemma 4.4 to show that a function $f \in \text{Aut}(I)$ is not an automorphism of R. As $f(\pi_{\leq i}(t))$ must be a tuple in $\pi_{\leq i}(R)$ according to Lemma 4.4, there must exist a tuple $s \in R$ such that $f(\pi_{\leq i}(t)) = \pi_{\leq i}(s)$. This tuple is chosen in the third loop of the algorithm. In addition, given that $f(\pi_{\leq i+1}(t)) \notin \pi_{\leq i+1}(R)$ according to Lemma 4.4, then it must be the case that $f(\pi_{i+1}(t)) \neq \pi_{i+1}(s)$. But, in fact, for every tuple $r \in R$ such that $\pi_{\leq i}(r) = \pi_{\leq i}(s)$, it must be the case that $f(\pi_{i+1}(t)) \neq \pi_{i+1}(r)$; otherwise, $f(\pi_{\leq i+1}(t))$ would be a tuple in $\pi_{\leq i+1}(R)$. The set BadElements contains all the possible values a for $f(\pi_{i+1}(t))$ that make $f(\pi_{i+1}(t))$ to satisfy this condition. Thus, in its innermost loop, Algorithm 1 picks a value $a \in$ BadElements, and makes the call `CheckForIso` (I, t, s, i, a) to check whether there exists an automorphism f of I such that $f(\pi_{\leq i}(t)) = \pi_{\leq i}(s)$ and $f(\pi_{i+1}(t)) = a$. If this is the case, then Algorithm 1 knows that $f \in \text{Aut}(I)$ and f is not an automorphism of R; thus, it returns **false**. Otherwise, after trying all possibilities for i, t, s and a, Algorithm 1 knows by Lemma 4.4 that every automorphism of I is an automorphism of R; thus, it returns **true**.

To check whether there exists an automorphism f of I such that $f(\pi_{\leq i}(t)) = \pi_{\leq i}(s)$ and $f(\pi_{i+1}(t)) = a$, function `CheckForIso` generalizes the approach given in Example 4.3, and uses an oracle for the Rel-Iso decision problem (in its penultimate line). More precisely, this function starts by creating two copies I_1 and I_2 of I. Then, it adds to I_1 the fresh facts $R_1(\pi_1(t)), \ldots, R_i(\pi_i(t)), R_a(\pi_{i+1}(t))$, and it adds to I_2 the fresh facts $R_1(\pi_1(s)), \ldots, R_i(\pi_i(s)), R_a(a)$. Finally, it calls the oracle to verify whether there exists an isomorphism from I_1 to I_2, which represents an automorphism of I satisfying the aforementioned conditions, as it has to map $\pi_j(t)$ to $\pi_j(s)$ $(1 \leq j \leq i)$ and $\pi_{i+1}(t)$ to a.

ALGORITHM 1: Algorithm for Deciding First-Order Logic Definability

Input: Relational instance I, relation R with arity$(R) = n$.
Output: **true** if $\mathbf{adom}(R) \subseteq \mathbf{adom}(I)$ and every automorphism of I is also an automorphism of R, and **false** otherwise.

```
if adom(R) ⊈ adom(I) then
 | return false
end
for i = 0 to n − 1 do
 | foreach t ∈ R do
 | | foreach s ∈ R do
 | | | BadElements ← {a ∈ adom(I) | ∀r ∈ R : if π≤i(r) = π≤i(s), then πi+1(r) ≠ a};
 | | | foreach a ∈ BadElements do
 | | | | if CheckForIso(I, t, s, i, a) then
 | | | | | return false
 | | | | end
 | | | end
 | | end
 | end
end
return true
```

Function `CheckForIso`(I, t, s, i, a)
Input: Relational instance I, n-ary tuples t and s, values $i \in [0, n]$ and $a \in \mathbf{adom}(I)$.
Output: **true** if there exists an automorphism f of I such that $f(\pi_{\leq i}(t)) = \pi_{\leq i}(s)$ and $f(\pi_{i+1}(t)) = a$, and **false** otherwise.

```
R ← Relational schema of I;
R* ← R ∪ {R1, ..., Ri, Ra}, where each Rj (1 ≤ j ≤ i) and Ra are fresh unary relation names;
I1 ← empty instance of R*;
I2 ← empty instance of R*;
foreach R ∈ R do
 | R^I1 ← R^I;
 | R^I2 ← R^I;
end
for j = 1 to i do
 | Rj^I1 ← {(πj(t))};
 | Rj^I2 ← {(πj(s))};
end
Ra^I1 ← {(πi+1(t))};
Ra^I2 ← {(a)};
if there exists an isomorphism from I1 to I2 (i.e., (I1, I2) ∈ Rel-Iso) then return true;
else return false;
```

Example 4.5. Continuing with Examples 2.1 and 3.1, now consider the definability problem for the pair (I, T), where I is defined as in Example 2.1 and T is the following relation:

T		
John	Dana	John
Ada	Dana	John

In this case, for $i = 0$, the algorithm will not find any automorphism of I that fails to be an automorphism of $\pi_1(T)$. In fact, the only nontrivial automorphism of I is the one that maps John $\rightarrow$ Ada, Ada $\rightarrow$ John, Dana $\rightarrow$ Dana, and Peter $\rightarrow$ Peter, and this one maps T correctly up to column $i = 1$. Similarly, for $i = 1$, we have that the only nontrivial automorphism of I is also an automorphism of $\pi_{\leq 2}(T)$; thus, again, the algorithm will not find the witness automorphism. For value $i = 2$, consider the iteration step at which $t = (\text{John}, \text{Dana}, \text{John})$ and $s = (\text{Ada}, \text{Dana}, \text{John})$. Then, we have that

$$\text{BadElements} = \{\text{Ada}, \text{Dana}, \text{Peter}\}.$$

We now iterate over the elements of BadElements. For $a = \text{Ada}$, we build instances I_1 and I_2, as follows. For I_1, we have that $\text{Person}^{I_1} = \text{Person}^{I}$ and $\text{Knows}^{I_1} = \text{Knows}^{I}$, and we add the fresh facts:

$\mathrm{T}_1^{I_1}$	$\mathrm{T}_2^{I_1}$	$\mathrm{T}_a^{I_1}$
John $(= \pi_1(t))$	Dana $(= \pi_2(t))$	John $(= \pi_3(t))$

For I_2, we have that $\text{Person}^{I_2} = \text{Person}^{I}$ and $\text{Knows}^{I_2} = \text{Knows}^{I}$, and that

$\mathrm{T}_1^{I_2}$	$\mathrm{T}_2^{I_2}$	$\mathrm{T}_a^{I_1}$
Ada $(= \pi_1(s))$	Dana $(= \pi_2(s))$	Ada $(= a)$

Then, we have that I_1 and I_2 are, in fact, isomorphic, whereby the Rel-Iso will return **true**. Therefore, as a witness has been found, the algorithm returns **false**.

Algorithm 1 runs in polynomial time in the size of the input, assuming that every call to the subroutine for the Rel-Iso decision problem takes constant time (i.e., assuming that Algorithm 1 has access to an oracle for the Rel-Iso decision problem). More precisely, let $|S|$ be the number of elements in a set S, and recall that $n = \text{arity}(R)$. Then, the outer loops of Algorithm 1 complete at most $|R|^2 \times n$ iterations. For each of these iterations, the set BadElements is computed in polynomial time, as at most $|\mathbf{adom}(I)|$ candidate elements a are tested, in which case, for each element a, the condition defining the set BadElements can be checked in polynomial time on $|R|$, $i \leq n$ and $|\mathbf{adom}(I)|$. Moreover, as to the subroutine CheckForIso, it builds the relational instances I_1 and I_2 in polynomial time as well.

From this discussion, the fact that Rel-Iso $\in$ **GI** and the transitivity of polynomial-time Turing reductions, we conclude that FO-Def $\leq_T^p$ Graph-Iso, whereby FO-Def $\in$ **GI**.

We will now show that FO-Def is **GI**-hard by showing that Aut-1-AFP $\leq_T^p$ FO-Def (we actually show a many-to-one reduction to the complement of FO-Def, which is a stronger result than we need). Given a graph $G = (V, E)$ and a node $v \in V$, build a relational instance I with only one relation E copying the edge relation of G. Finally, build the relation R in the following way: $R^I = \{(v)\}$, that is, R has arity 1 and only contains one tuple with the distinguished node v. Note that, as built, an automorphism f of I that is not an automorphism of R will be such that $f(v) \neq v$. Thus, we have that $(G, v) \in$ Aut-1-AFP if and only if $(I, R) \notin$ FO-Def. Hence, given that (I, R) can be constructed in polynomial time from (G, v), we conclude that the problem Aut-1-AFP can be solved in polynomial time by using an oracle for FO-Def.

We therefore conclude that Aut-1-AFP $\leq_T^p$ FO-Def, whereby FO-Def is **GI**-complete. $\square$

5. PRACTICAL CONSIDERATIONS AND POSSIBLE EXTENSIONS

Having established the exact complexity of the first-order logic definability problem, we now turn to possible variations of the problem and practical considerations. The definability problem, while of great theoretical interest, should be considered in the broader context of database research. As was mentioned in the introduction, the definability problem provides a common basis for research in reverse engineering [Tran et al. 2009; Zhang et al. 2013], querying by example [Abouzied et al. 2013; Bonifati et al. 2014a], view definitions [Sarma et al. 2010], and so on. In fact, FO-Def may be interpreted as a basic query reverse-engineering scenario, in which a user who has access to a dataset and an answer relation needs to discover the query (first-order query, in this case) that produced such an answer over the data. A natural extension of this scenario is one in which we must fit several such examples source–target pairs [Fletcher et al. 2009], which we discuss in Section 5.1. Such scenarios find applications in areas such as schema matching and data integration [Bilke and Naumann 2005; Gottlob and Senellart 2010; Qian et al. 2012; ten Cate et al. 2013]. In each of these areas, it may be interesting to explore the consequences of the graph-isomorphism-based approach to the definability problems presented here.

In terms of practical implementations of FO-Def itself, an algorithm for FO-Def whose efficiency depends on an external subroutine for the graph isomorphism problem—a heavily studied problem in its own right—comes with several benefits for optimization. Not only can we now tap into the power of highly optimized graph-isomorphism [Read and Corneil 1977; Arvind and Torán 2005; Torán and Wagner 2009; Köbler et al. 1993; Piperno 2008; McKay and Piperno 2014; Babai 2015], we can also consider all restrictions on the input graphs that produce efficiently solvable versions of the graph-isomorphism problem, and inherit those benefits in our FO-Def implementations (e.g., see Grohe [2011, 2012]).

As a final consideration, in Section 5.2, we comment on the use of constants in the queries. Although we will see that unrestricted constants results in an uninteresting problem, a more restricted use of constants may have practical applications that make this case worth looking into.

5.1. The BP-Pairs Problem

Expanding on the definability problem as a reverse engineering scenario, where a query must be obtained to match a source-target (relational instance-relation) pair, the situation where several such pairs are given is represented by the following decision problem:

$$\text{BP-Pairs} = \{((S_1, T_1), \ldots, (S_k, T_k)) \mid S_1, T_1, \ldots S_k, T_k \text{ are relations and there exists a first-order query } Q \text{ such that for every } i \in [1, k] : Q(S_i) = T_i\}.$$

In Fletcher et al. [2009], it was shown that $\overline{\text{Graph-Iso}} \leq^p_m$ BP-Pairs, that is, there exists a polynomial-time many-to-one reduction from $\overline{\text{Graph-Iso}}$ to BP-Pairs (this was referred to as *cograph-isomorphism-hardness* in Fletcher et al. [2009]). Moreover, it was also shown in Fletcher et al. [2009] that BP-Pairs $\in$ **coNP**. A corollary of the first result is that BP-Pairs is **GI**-hard, as a many-to-one reduction also constitutes a Turing reduction (the **GI**-hardness of BP-Pairs can be alternatively derived using the results from Section 4). The key insight regarding this generalized version of the definability problem is its semantic characterization: an input $((S_1, T_1), \ldots, (S_k, T_k))$ is in BP-Pairs if and only if (i) for every $i \in [1, k]$, we have $\mathbf{adom}(T_i) \subseteq \mathbf{adom}(S_i)$; and (ii) for every $i, j \in [1, k]$, we have that if f is an isomorphism from S_i to S_j, then it is also an isomorphism from T_i to T_j [Fletcher et al. 2009].

Algorithm 1 can be adapted to solve this decision problem as well, leading to the following:

Theorem 5.1. BP-Pairs $\in$ **GI**.

The previous result, along with the **GI**-hardness of BP-Pairs, as proven in Fletcher et al. [2009], gives the following result:

Corollary 5.2. *BP-Pairs is* **GI**-*complete.*

The previous corollary establishes the exact complexity of BP-PAIRS, thus closes a problem that was left open in Fletcher et al. [2009].

In order to prove Theorem 5.1, consider the following extension of Lemma 4.4:

LEMMA 5.3. *Let S_1, S_2, T_1, T_2 be relations such that $\mathbf{adom}(T_i) \subseteq \mathbf{adom}(S_i)$ for $i \in [1, 2]$. Given an isomorphism f from S_1 to S_2, f is not an isomorphism from T_1 to T_2 if and only if there exists a tuple $t \in T_1$ and an integer $i \in [0, n-1]$ such that:*

(1) $f(\pi_{\leq i}(t)) \in \pi_{\leq i}(T_2)$,
(2) $f(\pi_{\leq i+1}(t)) \notin \pi_{\leq i+1}(T_2)$.

The proof of this lemma is very similar to that of Lemma 4.4, thus has been omitted. With this result, an algorithm analogous to that shown in Section 4.1 is used to prove Theorem 5.1.

5.2. Including Constants in the Definability Problem

As mentioned previously, FO-DEF considers the existence of a first-order logic query without constants. Let FO-DEF-CONST be the decision problem consisting of pairs (I, R) such that there exists a first-order query *with constants* Q such that $Q(I) = R$. Then, FO-DEF-CONST can be decided in polynomial time due to the fact that a pair (I, R) will be included in FO-DEF-CONST if and only if $\mathbf{adom}(R) \subseteq \mathbf{adom}(I)$. It is evident that this problem has become uninteresting, as a query can always be found with the sole exception that a first-order query may not introduce new constants into the answer. The actual reverse-engineered query Q such that $Q(I) = R$ is not very informative, though; given an input (I, R) such that $\mathbf{adom}(R) \subseteq \mathbf{adom}(I)$ and arity$(R) = n$, the proof of FO-DEF-CONST $\in \mathbf{P}$ constructs a query Q of the form $\{(x_1, \ldots, x_n) \mid \bigvee_{t \in R} Q_t(x_1, \ldots, x_n)\}$, where, for a tuple $t = (a_1, \ldots, a_n)$ in R, the query $Q_t(x_1, \ldots, x_n)$ is the expression $\bigwedge_{i \in [1,n]} x_i = a_i$. This query is fine tuned to the specific pair (I, R) and does not shed light on the original unknown query which might have produced this pair. In fact, this query becomes useless if some constants in the input (I, R) are renamed thus, it is an example of overfitting.

A more restricted, and useful, use of constants is formalized in the following problem: FO-DEF-CONST-S $= \{(I, R, C)\} \mid I$ is a relational instance, R is a relation, C is a set of constants, and there exists a first-order query Q, which may mention constants in C only, such that $Q(I) = R\}$. This extension of FO-DEF is **GI**-complete. To see this, note that FO-DEF $\leq_m^p$ FO-DEF-CONST-S is trivial (by setting $C = \emptyset$), and that FO-DEF-CONST-S $\leq_m^p$ FO-DEF admits a simple proof as well. On input (I, R, C) to FO-DEF-CONST-S, construct an instance (I', R') to FO-DEF by encoding the constants in C into the instance I', using singleton relations. More precisely, set $R' = R$ and let I' have all the relations in I plus a unary singleton relation $C_i = \{(c_i)\}$ for each constant $c_i \in C$. The previous arrangement for (I', R') allows constants to be referred to indirectly by using the expression $C_i(x)$ in a first-order query, as it will be true only when x is assigned to c_i.

As a more elaborate—and interesting—setting, consider the problem FO-DEF-CONST-$\leq$ $= \{(I, R, 0^n) \mid I$ is a relational instance, R is a relation, and n is a natural number, such that there exists a first-order query Q that mentions at most n distinct constants, and $Q(I) = R\}$. Note that the input n in FO-DEF-CONST-$\leq$ is encoded in unary as a string of 0s of length n. Although FO-DEF-CONST-$\leq$ remains **GI**-hard (set $n = 0$ in a Turing reduction), it is no longer obviously in **GI**. Actually, FO-DEF-CONST-$\leq$ $\in \mathbf{NP}^{\mathbf{GI}}$, as a nondeterministic polynomial-time algorithm, may guess a set C of n constants and use an oracle for the FO-DEF-CONST-S problem. The question remains, then, whether FO-DEF-CONST-$\leq$ is in **GI**.

Finally, consider the case in which the queries have access to a linear order over the constants in the relational instance, which exhibits underlying similarities to the unrestricted constants case FO-DEF-CONST. Formally, consider the problem FO-DEF-LIN $= \{(I, R) \mid I$ to be a relational instance having a binary relation $<$ which is a linear order over all elements in $\mathbf{adom}(I)$, R is a relation, and there exists a first-order query Q such that $Q(I) = R\}$. In the presence of the linear order, and using the semantic characterization, the only automorphism of I is the identity (i.e., the function $h(x) = x$), which is trivially also an automorphism of R. Hence, in this case, an algorithm must ensure only that $\mathbf{adom}(R) \subseteq \mathbf{adom}(I)$ to check whether $(I, R) \in$ FO-DEF-LIN, which may be completed in polynomial time. Therefore, FO-DEF-LIN $\in \mathbf{P}$ and, once again, the problem becomes trivial. Moreover, this is also an example of over-fitting, as every element in I can be identified by its position in the linear order, which is used as in the case of FO-DEF-CONST to define a query Q such that $Q(I) = R$.

6. CONCLUSIONS

The first-order logic definability problem, FO-DEF, and the generalized version, BP-PAIRS, have been found to be **GI**-complete, thus closing two open problems in the database area. Two fundamental corollaries of these results are that FO-DEF can be solved efficiently if the graph-isomorphism problem can be solved efficiently, and that FO-DEF is not **coNP**-complete unless the polynomial hierarchy collapses to the second level. The incremental approach taken by the polynomial-time algorithm for FO-DEF with an oracle for the graph-isomorphism problem may prove applicable to other scenarios as well, and deserves further investigation.

ACKNOWLEDGMENTS

The authors would like to thank Michael Benedikt for participating in the discussions that led to the results presented, and to Miguel Romero and Balder ten Cate for providing many valuable comments and suggestions.

REFERENCES

Scott Aaronson, Greg Kuperberg, and Christopher Granade. 2005. Complexity Zoo. Retrieved March 28, 2016 from https://complexityzoo.uwaterloo.ca.

Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of Databases*. Addison-Wesley, New York, NY.

Azza Abouzied, Dana Angluin, Christos H. Papadimitriou, Joseph M. Hellerstein, and Avi Silberschatz. 2013. Learning and verifying quantified Boolean queries by example. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'13)*, New York, NY, June 22–27, 2013. 49–60.

Timos Antonopoulos, Frank Neven, and Frédéric Servais. 2013. Definability problems for graph query languages. In *Proceedings of the Joint 2013 EDBT/ICDT Conferences (ICDT'13)*, Genoa, Italy, March 18–22, 2013. 141–152.

Sanjeev Arora and Boaz Barak. 2009. *Computational Complexity - A Modern Approach*. Cambridge University Press, New York, NY.

Vikraman Arvind and Jacobo Torán. 2005. Isomorphism testing: Perspective and open problems. *Bulletin of the EATCS* 86, 66–84.

László Babai. 2015. Graph isomorphism in quasipolynomial time. *CoRR* abs/1512.03547 (2015). http://arxiv.org/abs/1512.03547

François Bancilhon. 1978. On the completeness of query languages for relational data bases. In *Proceedings of the 7th Symposium of Mathematical Foundations of Computer Science 1978*, Zakopane, Poland, September 4–8, 1978 *(Lecture Notes in Computer Science)*, Józef Winkowski (Ed.), Vol. 64. Springer, Berlin, 112–123.

Alexander Bilke and Felix Naumann. 2005. Schema matching using duplicates. In *Proceedings of the 21st International Conference on Data Engineering (ICDE'05)*, 5–8 April 2005, Tokyo, Japan. 69–80.

Angela Bonifati, Radu Ciucanu, and Aurélien Lemay. 2015. Learning path queries on graph databases. In *Proceedings of the 18th International Conference on Extending Database Technology (EDBT'15)*, Brussels, Belgium, March 23–27, 2015. 109–120.

Angela Bonifati, Radu Ciucanu, Aurélien Lemay, and Slawek Staworko. 2014b. A paradigm for learning queries on big data. In *Proceedings of the 1st International Workshop on Bringing the Value of "Big Data" to Users (Data4U@VLDB'14)*, Hangzhou, China, September 1, 2014. 7.

Angela Bonifati, Radu Ciucanu, and Slawek Staworko. 2014a. Interactive inference of join queries. In *Proceedings of the 17th International Conference on Extending Database Technology (EDBT'14)*, Athens, Greece, March 24–28, 2014. 451–462.

Ashok K. Chandra and David Harel. 1980. Computable queries for relational data bases. *Journal of Computer and System Sciences* 21, 2, 156–178.

Sara Cohen and Yaacov Y. Weiss. 2013. Certain and possible XPath answers. In *Proceedings of the Joint 2013 EDBT/ICDT Conferences (ICDT'13)*, Genoa, Italy, March 18–22, 2013. 237–248.

Herbert B. Enderton. 1972. *A Mathematical Introduction to Logic*. Academic Press, New York.

Flavio Antonio Ferrarotti, Alejandra Lorena Paoletti, and José M. Turull Torres. 2009. First-order types and redundant relations in relational databases. In *Proceedings of Advances in Conceptual Modeling - Challenging Perspectives, ER 2009 Workshops*, Gramado, Brazil, November 9–12, 2009. 65–74.

George H. L. Fletcher, Marc Gyssens, Jan Paredaens, and Dirk Van Gucht. 2009. On the expressive power of the relational algebra on finite sets of relation pairs. *IEEE Transactions on Knowledge and Data Engineering* 21, 6, 939–942.

George H. L. Fletcher, Marc Gyssens, Jan Paredaens, Dirk Van Gucht, and Yuqing Wu. 2015. Structural characterizations of the navigational expressiveness of relation algebras on a tree. *CoRR* abs/1502.03258 (2015).

Georg Gottlob and Pierre Senellart. 2010. Schema mapping discovery from data instances. *Journal of the ACM* 57, 2.

Martin Grohe. 2011. From polynomial time queries to graph structure theory. *Communications of the ACM* 54, 6, 104–112.

Martin Grohe. 2012. Fixed-point definability and polynomial time on graphs with excluded minors. *Journal of the ACM* 59, 5, 27.

Marc Gyssens, Jan Paredaens, and Dirk Van Gucht. 1989. A uniform approach toward handling atomic and structured information in the nested relational database model. *Journal of the ACM* 36, 4, 790–825.

Marc Gyssens, Jan Paredaens, Dirk Van Gucht, and George H. L. Fletcher. 2006. Structural characterizations of the semantics of XPath as navigation tool on a document. In *Proceedings of the 25th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, June 26–28, 2006, Chicago, IL, 318–327.

Lane A. Hemaspaandra. 1993. Lowness: A yardstick for NP-P. *SIGACT News* 24, 2, 10–14.

Peter Jeavons, David A. Cohen, and Marc Gyssens. 1999. How to determine the expressive power of constraints. *Constraints* 4, 2, 113–131.

Johannes Köbler, Uwe Schöning, and Jacobo Toran. 1993. *The Graph Isomorphism Problem: Its Structural Complexity*. Springer.

Anna Lubiw. 1981. Some NP-complete problems similar to graph isomorphism. *SIAM Journal on Computing* 10, 1, 11–21.

Brendan D. McKay and Adolfo Piperno. 2014. Practical graph isomorphism, II. *Journal of Symbolic Computation* 60, 94–112.

Jan Paredaens. 1978. On the expressive power of the relational algebra. *Information Processing Letters* 7, 2, 107–111.

Adolfo Piperno. 2008. Search space contraction in canonical labeling of graphs (preliminary version). *CoRR* abs/0804.4881 (2008). http://arxiv.org/abs/0804.4881

Li Qian, Michael J. Cafarella, and H. V. Jagadish. 2012. Sample-driven schema mapping. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'12)*, Scottsdale, AZ, May 20–24, 2012. 73–84.

Ronald C. Read and Derek G. Corneil. 1977. The graph isomorphism disease. *Journal of Graph Theory* 1, 4, 339–363.

Anish Das Sarma, Aditya G. Parameswaran, Hector Garcia-Molina, and Jennifer Widom. 2010. Synthesizing view definitions from data. In *Proceedings of the 13th International Conference on Database Theory (ICDT'10)*, Lausanne, Switzerland, March 23–25, 2010. 89–103.

Uwe Schöning. 1983. A low and a high hierarchy within NP. *Journal of Computer and System Sciences* 27, 1, 14–28.

Uwe Schöning. 1988. Graph isomorphism is in the low hierarchy. *Journal of Computer and System Sciences* 37, 3, 312–323.

Slawek Staworko and Piotr Wieczorek. 2012. Learning twig and path queries. In *15th International Conference on Database Theory (ICDT'12)*, Berlin, Germany, March 26–29, 2012. 140–154.

Slawek Staworko and Piotr Wieczorek. 2015. Characterizing XML twig queries with examples. In *18th International Conference on Database Theory (ICDT'15)*, March 23–27, 2015, Brussels, Belgium. 144–160.

Larry J. Stockmeyer. 1976. The polynomial-time hierarchy. *Theory of Computing Sciences.* 3, 1, 1–22.

Balder ten Cate and Víctor Dalmau. 2015. The product homomorphism problem and applications. In *18th International Conference on Database Theory (ICDT'15)*, March 23–27, 2015, Brussels, Belgium. 161–176.

Balder ten Cate, Víctor Dalmau, and Phokion G. Kolaitis. 2013. Learning schema mappings. *ACM Transactions on Database Systems* 38, 4, 28.

Jacobo Torán and Fabian Wagner. 2009. The complexity of planar graph isomorphism. *Bulletin of the EATCS* 97, 60–82.

Quoc Trung Tran, Chee-Yong Chan, and Srinivasan Parthasarathy. 2009. Query by output. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'09)*, Providence, RI, June 29 - July 2, 2009. 535–548.

Jan Van den Bussche. 2001. Applications of Alfred Tarski's ideas in database theory. In *Computer Science Logic, 15th International Workshop (CSL'01). Proceedings of the 10th Annual Conference of the EACSL*, Paris, France, September 10–13, 2001. 20–37.

Dirk Van Gucht. 1987. On the expressive power of the extended relational algebra for the unnormalized relational model. In *Proceedings of the 6th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, March 23–25, 1987, San Diego, California. 302–312.

Dirk Van Gucht. 2009. BP-completeness. In *Encyclopedia of Database Systems*. 265–266.

Ross Willard. 2010. Testing expressibility is hard. In *Proceedings of the 16th International Conference on Principles and Practice of Constraint Programming (CP'10)*, St. Andrews, Scotland, UK, September 6–10, 2010. 9–23.

V. N. Zemlyachenko, N. M. Korneenko, and R. I. Tyshkevich. 1985. Graph isomorphism problem. *Journal of Soviet Mathematics* 29, 4.

Meihui Zhang, Hazem Elmeleegy, Cecilia M. Procopiuc, and Divesh Srivastava. 2013. Reverse engineering complex join queries. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'13)*, New York, NY, June 22–27, 2013. 809–820.

Received May 2015; revised October 2015; accepted January 2016