What is View-Based Query Rewriting? (Position Paper)

Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini
Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, I-00198 Roma, Italy
lastname@dis.uniroma1.it

Moshe Y. Vardi
Department of Computer Science
Rice University, P.O. Box 1892
Houston, TX 77251-1892, U.S.A.
vardi@cs.rice.edu

Abstract

View-based query processing requires to answer a query posed to a database only on the basis of the information on a set of views, which are again queries over the same database. This problem is relevant in many aspects of database management, and has been addressed by means of two basic approaches, namely, query rewriting and query answering. In the former approach, one tries to compute a rewriting of the query in terms of the views, whereas in the latter, one aims at directly answering the query based on the view extensions. Based on recent results, we first show that already for very simple query languages, a rewriting is in general a coNP function wrt to the size of view extensions. Hence, the problem arises of characterizing which instances of the problem admit a rewriting that is PTIME. However, a tight connection between view-based query answering and constraint-satisfaction problems, allows us to show that the above characterization is going to be difficult.

1 Introduction

Several recent papers in the literature show that the problem of view-based query processing [25, 2] is relevant in many aspects of database management, including query optimization, data warehousing, data integration, and query answering with incomplete information. Informally speaking, the problem requires to answer a query posed to a database only on the basis of the information on a set of views, which are again queries over the same database. In query optimization, the problem is relevant because using the views may speed up query processing. In data integration, the views represent the only information sources accessible to answer a query. A data warehouse can be seen as a set of materialized views, and, therefore, query processing reduces to view-based query answering. Finally, since the views provide partial knowledge on the database, view-based query processing can be seen as a special case of query answering with incomplete information.

There are two approaches to view-based query processing, called *query rewriting* and *query answering*, respectively. In the former approach, we are given a query and a set of view definitions, and the goal is to reformulate the query into an expression, the *rewriting*, that refers only to the views, and provides the answer to the query. Typically, the rewriting is formulated in the same language used for the query and the views but in the alphabet of the view names, rather than the alphabet of the database. In the latter approach, besides the query and the view definitions, we are also given the extensions of the views. The goal is to compute the set of tuples that are implied by these extensions, i.e., the set of tuples that are in the answer set of the query in all the databases that are consistent with the views.

Notice the difference between the two approaches. In query rewriting, query processing is divided in two steps, where the first re-expresses the query in terms of a given query language over the alphabet of the view names, and the second evaluates the rewriting over the view extensions. In query answering, we do not pose any limit to query processing, and the only goal is to compute the answer to the query by exploiting all possible information, in particular the view extensions.

In the last years a large number of results have been reported for both problems. View-based query rewriting and query answering have been studied under different assumptions on the form of the queries and views. For query rewriting see, e.g., [19, 20, 23, 24, 10, 3, 14, 5, 9], and for query answering see, e.g., [2, 13, 6, 7, 4].

Unfortunately, many of these papers do not distinguish between view-based query answering and view-based query rewriting, and give raise to a sort of confusion between the two notions. Part of the problem comes from the fact that when the query and the views are conjunctive queries, there are algorithms for computing the best possible rewriting as union of conjunctive queries and therefore is basically expressible in the same language as the original query and views. However, as we will see in this paper, for other query languages this is not the case. So, in spite of the large amount of work on the subject, the relationship between view-based query rewriting and view-based query answering is not completely clarified yet.

In this paper we focus on this relationship. Abstracting from the language used to express the rewriting, thus generalizing the notion of rewriting considered in the literature, we define a *rewriting* of a query with respect to a set of views as a function that, given the extensions of the views, returns a set of tuples that is contained in the answer set of the query in every database consistent with the views. We call the rewriting that returns precisely such set the *perfect* rewriting of the query wrt the views. Observe that, by evaluating the perfect rewriting over given view extensions, one obtains the same set of tuples provided by view-based query answering. Hence, the perfect rewriting is the best rewriting that one can obtain, given the available information on both the definitions and the extensions of the views.

An immediate consequence of the relationship between perfect rewriting and query answering is that the data complexity of evaluating the perfect rewriting over the view extensions is the same as the data complexity of answering queries using views.

Typically, one is interested in queries that can be evaluated in PTIME (i.e., are PTIME functions in data complexity), and hence we would like rewritings to be PTIME as well. For queries and views that are conjunctive queries (without union), the perfect rewriting is a union of conjunctive queries and hence is PTIME [2]. By exploiting the results in [6, 8], we show that already for very simple query languages containing union the perfect rewriting is not PTIME in general. Hence, for such languages it would be interesting to characterize which instances of query rewriting admit a perfect rewriting that is PTIME. However, by establishing a tight connection between view-based query answering and constraint-satisfaction problems (CSP), we argue that this is going to be difficult due to its connection with a longstanding open problem for CSP [16, 11].

2 View-based query processing

Let us introduce the problem of view-based query answering [2, 13, 18, 6]. Consider a database that is accessible only through a set $\mathcal{V} = \{V_1, \ldots, V_k\}$ of views, and suppose we want to answer a query only on the basis of our knowledge on the views. Specifically, associated to each view V_i we have:

- its definition $\mathit{def}(V_i)$ in terms of a query over the alphabet Σ;
- information about its extension in terms of a set $\mathit{ext}(V_i)$ of tuples of objects[1].

We denote $(\mathit{def}(V_1), \ldots, \mathit{def}(V_k))$ by $\mathit{def}(\mathcal{V})$, $(\mathit{ext}(V_1), \ldots, \mathit{ext}(V_k))$ by $\mathit{ext}(\mathcal{V})$, and the set of objects appearing in $\mathit{ext}(\mathcal{V})$ by $\mathcal{D}_{\mathcal{V}}$.

Let Q be a query and DB a database. We denote by $\mathit{ans}(Q, DB)$ the *answer set* of Q over DB. We say that a database DB is *consistent with the views* $\mathcal{V}$ if $\mathit{ext}(V_i) \subseteq \mathit{ans}(\mathit{def}(V_i), DB)$, for each $V_i \in \mathcal{V}$. The *certain answer set of* Q *wrt the views* $\mathcal{V}$ is the set $\mathit{cert}(Q, \mathcal{V}) \subseteq \mathcal{D}_{\mathcal{V}}^n$ of n-tuples (where n is the arity of Q) such that $t \in \mathit{cert}(Q, \mathcal{V})$ if and only if $t \in \mathit{ans}(Q, DB)$, for every database DB that is consistent with $\mathcal{V}$.

We first introduce the decision problem version of view-based query answering, called *view-based boolean query answering*, as follows. Given

- a set $\mathcal{V}$ of views, their definitions $\mathit{def}(\mathcal{V})$, and extensions $\mathit{ext}(\mathcal{V})$,
- a query Q of arity n,
- a tuple of objects $t \in \mathcal{D}_{\mathcal{V}}^n$,

decide whether $t \in \mathit{cert}(Q, \mathcal{V})$. The problem of *view-based query answering* differs from its decision problem version, in that it does not have a tuple of objects as inputs, and aims at computing the whole $\mathit{cert}(Q, \mathcal{V})$.

The definition of view-based query answering given above reflects two implicit assumptions. (i) The views are *sound*, i.e., from the fact that a tuple t is in $\mathit{ext}(V_i)$ we can conclude that t is in $\mathit{ans}(\mathit{def}(V_i), DB)$, but not vice-versa. (ii) The domain is *open*, i.e., a database consistent with the views may contain additional objects that

[1] We assume that objects are represented by constants, and we adopt the *unique name assumption* [21], i.e., different constants denote different objects.

do not appear in the view extensions. Other assumptions about the accurateness of the knowledge on the objects of the database and the tuples satisfying the views, have been studied. For a discussion, see [2, 13, 6].

Next we focus on view-based query rewriting. An instance of *view-based query rewriting* is given by a query Q and a set $\mathcal{V}$ of views with definitions $def(\mathcal{V})$, and the goal is to compute a new query R over the symbols in $\mathcal{V}$ such that $ans(R, ext(\mathcal{V})) \subseteq cert(Q, \mathcal{V})$. In other words one tries to generate a new query over the symbols in $\mathcal{V}$ that approximates the answer to Q, when V_i is interpreted as $ext(V_i)$, for each $V_i \in \mathcal{V}$.

From an abstract point of view, a *rewriting of Q wrt $\mathcal{V}$* is a *function* that, given $ext(\mathcal{V})$, returns a set of tuples of objects that is contained in the certain answer set $cert(Q, \mathcal{V})$. The problem of *view-based query rewriting* is the one of computing one such function. The problem comes in different forms, depending of the properties that we require for the rewriting [19, 20, 10, 3, 5, 8]. In particular:

- It is sometimes interesting to consider rewritings that are expressible in a certain query language, e.g., Datalog.
- It is also interesting to consider rewritings belonging to a certain data complexity class, for example, polynomial time. A rewriting f belongs to a data complexity class $\mathcal{C}$ if the problem of deciding whether a tuple of objects is in $f(ext(\mathcal{V}))$ is in the class $\mathcal{C}$, where the complexity of the problem is measured with respect to the size of $ext(\mathcal{V})$.
- Finally, it is worth computing rewritings that are maximal in a certain class. A rewriting f of Q wrt $\mathcal{V}$ is *maximal in a class $\mathcal{C}$* if, for every rewriting $g \in \mathcal{C}$ of Q wrt $\mathcal{V}$, we have that $g(ext(\mathcal{V})) \subseteq f(ext(\mathcal{V}))$ for every $ext(\mathcal{V})$.

However, a question comes up: Which is the *best possible rewriting*? The answer to this question is: *the rewriting (or, any rewriting) P such that, for every $ext(\mathcal{V})$, we have that $ans(P, ext(\mathcal{V})) = cert(Q, \mathcal{V})$*. In other words, the best possible rewriting is the one that exploits the information on the views as much as possible, exactly like an algorithm for view-based query answering. We call any such rewriting *perfect rewriting* of Q wrt $\mathcal{V}$. Perfect rewritings should not be confused with exact rewritings. A rewriting R of Q wrt $\mathcal{V}$ is *exact* if the query R' obtained by substituting each view symbols with the corresponding definition, satisfies the following condition: for each database DB, $ans(R', DB) = ans(Q, DB)$.

Note that, there are Q and $\mathcal{V}$ such that no exact rewriting of Q wrt $\mathcal{V}$ exist, whereas the perfect rewriting always exists. In particular, a perfect rewriting can be obtained from any algorithm for view-based query answering. Indeed an algorithm for view-based query answering takes as input a query, a set of view definitions, and a set of view extensions, and computes the answer set of the query for every database that is consistent with the views. Hence, every algorithm for view-based query answering, instantiated on a query Q and view definitions $def(\mathcal{V})$, *is the perfect rewriting* of Q wrt $\mathcal{V}$. From this observation, one can conclude, for example, that the problem of view-based query rewriting raised in [3] has been solved. Indeed the algorithm presented in [4] directly yields the perfect rewriting for description logics knowledge bases. Obviously it remains open whether such a rewriting can be formulated in a conventional query language, such as disjunctive Datalog.

3 Hardness of view-based query processing

We consider a simple setting in which a database is constituted by a set of binary relations, and hence can be viewed as an edge-labeled graph. We use as query language *unions of path queries* (UPQs), defined as follows:

$$\begin{array}{rcl} Q & \longrightarrow & P \mid Q_1 \cup Q_2 \\ P & \longrightarrow & R \mid P_1 \circ P_2 \end{array}$$

where R denotes a (binary) database relation, P denotes a *path query*, which is a chaining of database relations, and Q denotes a union of path queries. Observe that such a language is a simplified form both of unions of conjunctive queries [25] and of regular path queries [1].

The problems of view-based query answering and query rewriting for UPQs are defined as in the previous section, considering that now Q and $def(V_1), \ldots, def(V_k)$ are UPQs over the alphabet Σ, while Q' is an UPQ over the alphabet $\mathcal{V}$.

Considering first view-based query answering, we observe that the complexity of the problem can be measured in three different ways [26]:

- *Data complexity*: as a function of the size of $ext(\mathcal{V})$.
- *Expression complexity*: as a function of the size of Q and of the expressions in $def(\mathcal{V})$.
- *Combined complexity*: as a function of the size of $ext(\mathcal{V})$, Q, and $def(\mathcal{V})$.

Here we are interested in data complexity.

The following theorem gives a lower bound for the data complexity of view-based query answering for UPQs.

Theorem 3.1 ([6]) *View-based boolean query answering for UPQs is coNP-hard in data complexity.*

Proof. The result follows from the lower bound for regular path queries in [6]. It is easy to see that the proof of that lower bound does not exploit reflexive transitive closure, and hence holds also for UPQs. □

In fact, being view-based query answering for regular path queries coNP-complete [6], we have that the problem is coNP-complete for UPQs.

Now, we have seen in the previous section that a perfect rewriting can be obtained from an algorithm for view-based query answering by fixing the query and the view definitions. Hence we have that the complexity of evaluating the perfect rewriting is the data complexity of view-based query answering.

We remind the reader that the *recognition problem* for a query Q is to check whether a certain tuple is in the answer of Q over a given database. Since Q can be regarded as a function, we say that Q is a coNP-hard function if the recognition problem for Q is coNP-hard. Now, by Theorem 3.1, we obtain the following result.

Theorem 3.2 *There is an UPQ Q and a set $\mathcal{V}$ of UPQ views such that the perfect rewriting of Q wrt $\mathcal{V}$ is a coNP-hard function.*

Typically, one is interested in queries that are PTIME functions. Hence, we would like rewritings to be PTIME as well. Unfortunately, even for such a simple language (containing union) as UPQs, by Theorem 3.2, the perfect rewritings are not PTIME in general. Hence it would be interesting to characterize which instances of query rewriting admit a perfect rewriting that is PTIME. Note, however, that finding such instances corresponds to finding those instances of view-based query answering that are PTIME in data complexity. Next we show that this is going to be difficult, by exhibiting a tight connection between view-based query answering and constraint satisfaction.

4 Constraint-satisfaction problems

A *constraint-satisfaction problem (CSP)* is traditionally defined in terms of a set of variables, a set of values, and a set of constraints, and asks whether there is an assignment of the variables with the values that satisfies the constraints. An elegant characterization of CSP can be given in terms of homomorphisms between relational structures [11].

A *vocabulary* is a set $V = \{R_1, \ldots, R_t\}$ of predicates, each with an associated arity. A *relational structure* $A = (\Delta^A, \cdot^A)$ over V is a *domain* Δ^A together with an *interpretation function* $\cdot^A$ that assigns to each predicate R_i a relation R_i^A of the appropriate arity over Δ^A. A *homomorphism* $h : A \to B$ between two relational structures A and B over the same vocabulary is a mapping $h : \Delta^A \to \Delta^B$ such that, if $(c_1, \ldots, c_n) \in R^A$, then $(h(c_1), \ldots, h(c_n)) \in R^B$, for every predicate R in the vocabulary.

Let $\mathcal{A}$ and $\mathcal{B}$ be two classes of finite relational structures. The *(uniform) constraint-satisfaction problem* $\mathrm{CSP}(\mathcal{A}, \mathcal{B})$ is the following decision problem: given a structure $A \in \mathcal{A}$ and a structure $B \in \mathcal{B}$ over the same vocabulary, is there a homomorphism $h : A \to B$? We denote such instance as $\mathrm{CSP}(A, B)$, and if such a homomorphism exists we say that $\mathrm{CSP}(A, B)$ is *satisfiable*. We also consider the special case where $\mathcal{B}$ consists of a single relational structure B and $\mathcal{A}$ is the set of all relational structures over the vocabulary of B, and denote it by $\mathrm{CSP}(B)$. Such problem is a (special case of) *non-uniform* constraint-satisfaction problem, i.e., with B fixed, the input is just a structure $A \in \mathcal{A}$. In the case where we take the relational structures to be (directed) graphs, CSP corresponds to *directed-graph homomorphism*. Since general CSP is polynomially equivalent to directed-graph homomorphism [11], that is, for each structure B there is a directed graph G_B such that $\mathrm{CSP}(B)$ is polynomially equivalent to $\mathrm{CSP}(G_B)$, we restrict attention without loss of generality to CSP over directed graphs, unless explicitly stated otherwise.

From the very definition of CSP it follows directly that every CSP($\mathcal{A}, \mathcal{B}$) problem is in NP. In general, the complexity of a non-uniform constraint-satisfaction problem CSP(B) depends on B. For example, CSP(K_2), is the *Two-Colorability Problem*, while CSP(K_3) is the *Three-Colorability Problem* (K_n is the n-node complete graph); the former is in PTIME, while the latter is NP-complete. In some cases, e.g., when the domain of B has at most two elements or when B is an undirected graph, it is known that CSP(B) is either in PTIME or NP-complete [22, 15]. The Dichotomy Conjecture states that this holds for every structure B [11]. (Recall that if PTIME is different than NP then there are problems that are neither in PTIME nor NP-complete [17].) It is an open problem whether the Dichotomy Conjecture holds. A related open question is that of characterizing the structures B for which CSP(B) is in PTIME [11].

5 CSP and view-based query processing

We establish a tight relationship between constraint-satisfaction problems and view-based query answering and query rewriting. We show first that every CSP is polynomially reducible to view-based boolean query answering.

Theorem 5.1 *Let B be a directed graph. There exists an UPQ Q and UPQ views $\mathcal{V}$ with definitions $def(\mathcal{V})$ such that the following holds: for every directed graph A, there are extensions $ext(\mathcal{V})$ and objects c, d such that $(c, d) \notin cert(Q, \mathcal{V})$ if and only if CSP(A, B) is satisfiable.*

Proof (sketch). Let $A = (N_A, E_A)$ and $B = (N_B, E_B)$. We define an instance of view-based boolean query answering as follows:

- The alphabet is $\Sigma = \Sigma_N \cup \Sigma_E$, where $\Sigma_N = \{S_x \mid x \in N_B\} \cup \{F_x \mid x \in N_B\}$ and $\Sigma_E = \{R_{x,y} \mid (x, y) \in E_B\}$.
- The set of objects in the view extensions is $\mathcal{D}_\mathcal{V} = N_A \cup \{c, d\}$, where c, d are two symbols not in N_A.
- The views are V_s, V_f, and V_A with

$$\begin{array}{rclcrcl} def(V_s) & = & \bigcup_{x \in N_B} S_x & \qquad & ext(V_s) & = & \{(c, a) \mid a \in N_A\} \\ def(V_f) & = & \bigcup_{x \in N_B} F_x & & ext(V_f) & = & \{(a, d) \mid a \in N_A\} \\ def(V_A) & = & \bigcup_{(x,y) \in E_B} R_{x,y} & & ext(V_A) & = & E_A \end{array}$$

  Intuitively, the extension of V_A represents A, while V_s and V_f are used to connect c and d to all nodes of A, using respectively the "start" relations S_x and "final" relations F_x.

- The query is

$$Q = \bigcup_{x,y \in N_B,\, x \neq y} S_x \circ F_y \quad \cup \quad \bigcup_{\substack{x \in N_B \\ y \neq x,\, (y,z) \in E_B}} S_x \circ R_{y,z} \circ F_z \quad \cup \quad \bigcup_{\substack{x \in N_B \\ (x,y) \in E_B,\, z \in N_B \setminus \{y\}}} S_x \circ R_{x,y} \circ F_z$$

It is possible to show that there is a homomorphism from A to B if and only if $(c, d) \notin cert(Q, \mathcal{V})$. $\square$

Theorem 5.1 exhibits a strong connection between CSP and view-based query answering. If we take as input for the CSP both A and B (i.e., we consider uniform CSP), this corresponds to taking as input both the extension and the view and query definitions. Since the reduction in the proof above is polynomial, we get the following corollary.

Corollary 5.2 *Every uniform CSP is polynomially reducible to view-based boolean query answering for UPQs.*

Moreover, in the reduction of Theorem 5.1, the query and the view definitions depend only on graph B, and only the view extensions depend on graph A. Hence, the theorem shows also that non-uniform CSP, where B (corresponding to query and view definitions) is fixed and the input is only A (corresponding to the view extensions), can be polynomially reduced to checking whether a pair of objects is in the answer set of the perfect rewritings wrt the view extensions. More precisely we have the following corollary:

Corollary 5.3 *Every non-uniform CSP is polynomially reducible to the recognition problem for perfect rewritings for UPQs.*

Observe that the difference between view-based boolean query answering and the recognition problem for perfect rewritings is that, in the first case the input includes the query, the view definitions, and the view extensions and a pair of objects, while in the latter case one already has a perfect rewriting and wants to check whether a given pair is in the answer set wrt the view extensions.

The theorem and the corollaries above indicate that there is a close relationship between the problem of characterizing the instances of query rewriting that admit a perfect rewriting that is in PTIME and the problem of characterizing the instances of CSP that are in PTIME. In particular as a consequence of Theorem 5.1, if we had a method to decide whether an instance of query rewriting admits a perfect rewriting that is PTIME, we would then be able to single out an interesting class of instances of non-uniform CSP that are in PTIME. However, as discussed in [16, 11], characterizing those instances of non-uniform CSP that are in PTIME is a longstanding open problem that appears to be difficult to solve. This is an indication of the difficulty of isolating those instances of the rewriting problem which admit a PTIME perfect rewriting.

6 Conclusions

We have set up a framework that clarifies the relationships between view-based query rewriting and view-based query answering. Based on such a framework, we have first shown that the perfect rewriting is in general a coNP function wrt to the size of view extensions. We have then turned our attention to the problem of characterizing which instances of query rewriting admit a rewriting that is PTIME. Based on a tight connection between view-based query answering and constraint-satisfaction problems, we have shown that the above characterization is going to be difficult.

The discussion above shows that in general there is a tradeoff between completeness of a rewriting and the efficiency of using the rewriting to compute the answers to the query. To retain efficiency, one has in general to give up the perfectness of the rewriting, and adopt weaker notions of completeness. For example, one can fix a priori the language of the rewriting (such language should have PTIME data complexity) and find a maximal rewriting expressible in such a language, possibly verifying whether it is an exact rewriting.

References

[1] Serge Abiteboul. Querying semi-structured data. In *Proc. of the 6th Int. Conf. on Database Theory (ICDT'97)*, pages 1–18, 1997.

[2] Serge Abiteboul and Oliver Duschka. Complexity of answering queries using materialized views. In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS'98)*, pages 254–265, 1998.

[3] Catriel Beeri, Alon Y. Levy, and Marie-Christine Rousset. Rewriting queries using views in description logics. In *Proc. of the 16th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS'97)*, pages 99–108, 1997.

[4] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Answering queries using views over description logics knowledge bases. In *Proc. of the 17th Nat. Conf. on Artificial Intelligence (AAAI 2000)*, 2000. To appear.

[5] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Rewriting of regular expressions and regular path queries. In *Proc. of the 18th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS'99)*, pages 194–204, 1999.

[6] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Answering regular path queries using views. In *Proc. of the 16th IEEE Int. Conf. on Data Engineering (ICDE 2000)*, pages 389–398, 2000.

[7] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Query processing using views for regular path queries with inverse. In *Proc. of the*

19th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS 2000), pages 58–66, 2000.

[8] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. View-based query processing and constraint satisfaction. In *Proc. of the 15th IEEE Sym. on Logic in Computer Science (LICS 2000)*, 2000.

[9] Sara Cohen, Werner Nutt, and Alexander Serebrenik. Rewriting aggregate queries using views. In *Proc. of the 18th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS'99)*, pages 155–166, 1999.

[10] Oliver M. Duschka and Michael R. Genesereth. Answering recursive queries using views. In *Proc. of the 16th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS'97)*, pages 109–116, 1997.

[11] Tomás Feder and Moshe Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction. *SIAM J. on Computing*, 28:57–104, 1999.

[12] Matthew L. Ginsberg, editor. *Readings in Nonmonotonic Reasoning.* Morgan Kaufmann, Los Altos, 1987.

[13] Gösta Grahne and Alberto O. Mendelzon. Tableau techniques for querying information sources through global schemas. In *Proc. of the 7th Int. Conf. on Database Theory (ICDT'99)*, volume 1540 of *Lecture Notes in Computer Science*, pages 332–347. Springer-Verlag, 1999.

[14] Jarek Gryz. Query folding with inclusion dependencies. In *Proc. of the 14th IEEE Int. Conf. on Data Engineering (ICDE'98)*, pages 126–133, 1998.

[15] P. Hell and J. Nešetřil. On the complexity of H-coloring. *J. of Combinatorial Theory, Series B*, 48:92–110, 1990.

[16] Phokion G. Kolaitis and Moshe Y. Vardi. Conjunctive-query containment and constraint satisfaction. In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS'98)*, pages 205–213, 1998.

[17] Richard E. Ladner. On the structure of polynomial time reducibility. *J. of the ACM*, 22:155–171, 1975.

[18] Alon Y. Levy. Obtaining complete answers from incomplete databases. In *Proc. of the 22nd Int. Conf. on Very Large Data Bases (VLDB'96)*, pages 402–412, 1996.

[19] Alon Y. Levy, Alberto O. Mendelzon, Yehoshua Sagiv, and Divesh Srivastava. Answering queries using views. In *Proc. of the 14th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS'95)*, pages 95–104, 1995.

[20] Anand Rajaraman, Yehoshua Sagiv, and Jeffrey D. Ullman. Answering queries using templates with binding patterns. In *Proc. of the 14th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS'95)*, 1995.

[21] Raymond Reiter. On closed world data bases. In Hervé Gallaire and Jack Minker, editors, *Logic and Databases*, pages 119–140. Plenum Publ. Co., New York, 1978. Republished in [12].

[22] Thomas J. Schaefer. The complexity of satisfiability problems. In *Proc. of the 10th ACM Sym. on Theory of Computing (STOC'78)*, pages 216–226, 1978.

[23] D. Srivastava, S. Dar, H. V. Jagadish, and A. Levy. Answering queries with aggregation using views. In *Proc. of the 22nd Int. Conf. on Very Large Data Bases (VLDB'96)*, pages 318–329, 1996.

[24] O. G. Tsatalos, M. H. Solomon, and Y. E. Ioannidis. The GMAP: A versatile tool for phyisical data independence. *Very Large Database J.*, 5(2):101–118, 1996.

[25] Jeffrey D. Ullman. Information integration using logical views. In *Proc. of the 6th Int. Conf. on Database Theory (ICDT'97)*, volume 1186 of *Lecture Notes in Computer Science*, pages 19–40. Springer-Verlag, 1997.

[26] Moshe Y. Vardi. The complexity of relational query languages. In *Proc. of the 14th ACM SIGACT Sym. on Theory of Computing (STOC'82)*, pages 137–146, 1982.