Data Integration: A Theoretical Perspective

Maurizio Lenzerini
Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, I-00198 Roma, Italy
lenzerini@dis.uniroma1.it

ABSTRACT

Data integration is the problem of combining data residing at different sources, and providing the user with a unified view of these data. The problem of designing data integration systems is important in current real world applications, and is characterized by a number of issues that are interesting from a theoretical point of view. This document presents on overview of the material to be presented in a tutorial on data integration. The tutorial is focused on some of the theoretical issues that are relevant for data integration. Special attention will be devoted to the following aspects: modeling a data integration application, processing queries in data integration, dealing with inconsistent data sources, and reasoning on queries.

1. INTRODUCTION

Data integration is the problem of combining data residing at different sources, and providing the user with a unified view of these data [60, 61, 89]. The problem of designing data integration systems is important in current real world applications, and is characterized by a number of issues that are interesting from a theoretical point of view. This tutorial is focused on some of these theoretical issues, with special emphasis on the following topics.

The data integration systems we are interested in this work are characterized by an architecture based on a global schema and a set of sources. The sources contain the real data, while the global schema provides a reconciled, integrated, and virtual view of the underlying sources. Modeling the relation between the sources and the global schema is therefore a crucial aspect. Two basic approaches have been proposed to this purpose. The first approach, called global-as-view, requires that the global schema is expressed in terms of the data sources. The second approach, called local-as-view, requires the global schema to be specified independently from the sources, and the relationships between the global schema and the sources are established by defining every source as a view over the global schema. Our goal is to discuss the characteristics of the two modeling mechanisms, and to mention other possible approaches.

Irrespectively of the method used for the specification of the mapping between the global schema and the sources, one basic service provided by the data integration system is to answer queries posed in terms of the global schema. Given the architecture of the system, query processing in data integration requires a reformulation step: the query over the global schema has to be reformulated in terms of a set of queries over the sources. In this tutorial, such a reformulation problem will be analyzed for both the case of local-as-view, and the case of global-as-view mappings. A main theme will be the strong relationship between query processing in data integration and the problem of query answering with incomplete information.

Since sources are in general autonomous, in many real-world applications the problem arises of mutually inconsistent data sources. In practice, this problem is generally dealt with by means of suitable transformation and cleaning procedures applied to data retrieved from the sources. In this tutorial, we address this issue from a more theoretical perspective.

Finally, there are several tasks in the operation of a data integration system where the problem of reasoning on queries (e.g., checking whether two queries are equivalent) is relevant. Indeed, query containment is one of the basic problems in database theory, and we will discuss several notions generalizing this problem to a data integration setting.

The paper is organized as follows. Section 2 presents our formalization of a data integration system. In Section 3 we discuss the various approaches to modeling. Sections 4 and 5 present an overview of the methods for processing queries in the local-as-view and in the global-as-view approach, respectively. Section 6 discusses the problem of dealing with inconsistent sources. Section 7 provides an overview on the problem of reasoning on queries. Finally, Section 8 concludes the paper by mentioning some open problems, and several research issues related to data integration that are not addressed in the tutorial.

2. DATA INTEGRATION FRAMEWORK

In this section we set up a logical framework for data integration. We restrict our attention to data integration systems

.

based on a so-called global schema (or, mediated schema). In other words, we refer to data integration systems whose aim is combining the data residing at different sources, and providing the user with a unified view of these data. Such a unified view is represented by the global schema, and provides a reconciled view of all data, which can be queried by the user. Obviously, one of the main task in the design of a data integration system is to establish the mapping between the sources and the global schema, and such a mapping should be suitably taken into account in formalizing a data integration system.

It follows that the main components of a data integration system are the global schema, the sources, and the mapping. Thus, we formalize a *data integration system* $\mathcal{I}$ in terms of a triple $\langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, where

- $\mathcal{G}$ is the *global schema*, expressed in a language $\mathcal{L}_\mathcal{G}$ over an alphabet $\mathcal{A}_\mathcal{G}$. The alphabet comprises a symbol for each element of $\mathcal{G}$ (i.e., relation if $\mathcal{G}$ is relational, class if $\mathcal{G}$ is object-oriented, etc.).
- $\mathcal{S}$ is the *source schema*, expressed in a language $\mathcal{L}_\mathcal{S}$ over an alphabet $\mathcal{A}_\mathcal{S}$. The alphabet $\mathcal{A}_\mathcal{S}$ includes a symbol for each element of the sources.
- $\mathcal{M}$ is the *mapping* between $\mathcal{G}$ and $\mathcal{S}$, constituted by a set of *assertions* of the forms

$$q_\mathcal{S} \rightsquigarrow q_\mathcal{G},$$
$$q_\mathcal{G} \rightsquigarrow q_\mathcal{S}$$

  where $q_\mathcal{S}$ and $q_\mathcal{G}$ are two queries of the same arity, respectively over the source schema $\mathcal{S}$, and over the global schema $\mathcal{G}$. Queries $q_\mathcal{S}$ are expressed in a query language $\mathcal{L}_{\mathcal{M},\mathcal{S}}$ over the alphabet $\mathcal{A}_\mathcal{S}$, and queries $q_\mathcal{G}$ are expressed in a query language $\mathcal{L}_{\mathcal{M},\mathcal{G}}$ over the alphabet $\mathcal{A}_\mathcal{G}$. Intuitively, an assertion $q_\mathcal{S} \rightsquigarrow q_\mathcal{G}$ specifies that the concept represented by the query $q_\mathcal{S}$ over the sources corresponds to the concept in the global schema represented by the query $q_\mathcal{G}$ (similarly for an assertion of type $q_\mathcal{G} \rightsquigarrow q_\mathcal{S}$). We will discuss several ways to make this intuition precise in the following sections.

Intuitively, the source schema describes the structure of the sources, where the real data are, while the global schema provides a reconciled, integrated, and virtual view of the underlying sources. The assertions in the mapping establish the connection between the elements of the global schema and those of the source schema.

Queries to $\mathcal{I}$ are posed in terms of the global schema $\mathcal{G}$, and are expressed in a query language $\mathcal{L}_\mathcal{Q}$ over the alphabet $\mathcal{A}_\mathcal{G}$. A query is intended to provide the specification of which data to extract from the virtual database represented by the integration system.

The above definition of data integration system is general enough to capture virtually all approaches in the literature. Obviously, the nature of a specific approach depends on the characteristics of the mapping, and on the expressive power of the various schema and query languages. For example, the language $\mathcal{L}_\mathcal{G}$ may be very simple (basically allowing the definition of a set of relations), or may allow for various forms of integrity constraints to be expressed over the symbols of $\mathcal{A}_\mathcal{G}$. Analogously, the type (e.g., relational, semistructured, etc.) and the expressive power of $\mathcal{L}_\mathcal{S}$ varies from one approach to another.

We now specify the semantics of a data integration system. In what follows, a database (DB) for a schema $\mathcal{T}$ is simply a set of collection of sets, one for each symbol in the alphabet of $\mathcal{T}$ (e.g., one relation for every relation schema of $\mathcal{T}$, if $\mathcal{T}$ is relational, or one set of objects for each class of $\mathcal{T}$, if $\mathcal{T}$ is object-oriented, etc.). We also make a simplifying assumption on the domain for the various sets. In particular, we assume that the structures constituting the databases involved in our framework (both the global database and the source databases) are defined over a fixed domain Γ.

In order to assign semantics to a data integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, we start by considering a *source database* for $\mathcal{I}$, i.e., a database $\mathcal{D}$ that conforms to the source schema $\mathcal{S}$ and satisfies all constraints in $\mathcal{S}$. Based on $\mathcal{D}$, we now specify which is the information content of the global schema $\mathcal{G}$. We call *global database* for $\mathcal{I}$ any database for $\mathcal{G}$. A global database $\mathcal{B}$ for $\mathcal{I}$ is said to be *legal with respect to* $\mathcal{D}$, if:

- $\mathcal{B}$ is legal with respect to $\mathcal{G}$, i.e., $\mathcal{B}$ satisfies all the constraints of $\mathcal{G}$;
- $\mathcal{B}$ satisfies the mapping $\mathcal{M}$ with respect to $\mathcal{D}$.

The notion of $\mathcal{B}$ satisfying the mapping $\mathcal{M}$ with respect to $\mathcal{D}$ depends on how to interpret the assertions in the mapping. We will see in the next section that several approaches are conceivable. Here, we simply note that, no matter which is the interpretation of the mapping, in general, several global databases exist that are legal for $\mathcal{I}$ with respect to $\mathcal{D}$. This observation motivates the relationship between data integration and databases with incomplete information [91], which will be discussed in several ways later on in the paper.

Finally, we specify the semantics of queries posed to a data integration system. As we said before, such queries are expressed in terms of the symbols in the global schema of $\mathcal{I}$. In general, if q is a query of arity n and $\mathcal{DB}$ is a database, we denote with $q^{\mathcal{DB}}$ the set of tuples (of arity n) in $\mathcal{DB}$ that satisfy q.

Given a source database $\mathcal{D}$ for $\mathcal{I}$, the answer $q^{\mathcal{I},\mathcal{D}}$ to a query q in $\mathcal{I}$ with respect to $\mathcal{D}$, is the set of tuples t of objects in Γ such that $t \in q^{\mathcal{B}}$ for *every* global database $\mathcal{B}$ that is legal for $\mathcal{I}$ with respect to $\mathcal{D}$. The set $q^{\mathcal{I},\mathcal{D}}$ is called the set of *certain answers* to q in $\mathcal{I}$ with respect to $\mathcal{D}$.

Note that, from the point of view of logic, finding certain answers is a logical implication problem: check whether it logically follows from the information on the sources that t satisfies the query. The dual problem is also of interest: finding the so-called *possible answers* to q, i.e., checking whether $t \in q^{\mathcal{B}}$ for some global database $\mathcal{B}$ that is legal for $\mathcal{I}$ with respect to $\mathcal{D}$. Finding possible answers is a consistency problem: check whether assuming that t is in the answer set of q does not contradict the information on the sources.

3. MODELING

One of the most important aspects in the design of a data integration system is the specification of the correspondence between the data at the sources and those in the global schema. Such a correspondence is modeled through the notion of mapping as introduced in the previous section. It is exactly this correspondence that will determine how the queries posed to the system are answered.

In this section we discuss mappings which can be expressed in terms of first order logic assertions. Mappings going beyond first order logic are briefly discussed in Section 6.

Two basic approaches for specifying the mapping in a data integration system have been proposed in the literature, called *local-as-view* (LAV), and *global-as-view* (GAV), respectively [89, 60]. We discuss these approaches separately. We then end the section with a comparison of the two kinds of mapping.

3.1 Local as view

In a data integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ based on the LAV approach, the mapping $\mathcal{M}$ associates to each element s of the source schema $\mathcal{S}$ a query $q_\mathcal{G}$ over $\mathcal{G}$. In other words, the query language $\mathcal{L}_{\mathcal{M},\mathcal{S}}$ allows only expressions constituted by one symbol of the alphabet $\mathcal{A}_\mathcal{S}$. Therefore, a LAV mapping is a set of assertions, one for each element s of $\mathcal{S}$, of the form

$$s \leadsto q_\mathcal{G}$$

From the modeling point of view, the LAV approach is based on the idea that the content of each source s should be characterized in terms of a view $q_\mathcal{G}$ over the global schema. A notable case of this type is when the data integration system is based on an enterprise model, or an ontology [58]. This idea is effective whenever the data integration system is based on a global schema that is stable and well-established in the organization. Note that the LAV approach favors the extensibility of the system: adding a new source simply means enriching the mapping with a new assertion, without other changes.

To better characterize each source with respect to the global schema, several authors have proposed more sophisticated assertions in the LAV mapping, in particular with the goal of establishing the assumption holding for the various source extensions [1, 53, 65, 24]. Formally, this means that in the LAV mapping, a new specification, denoted $as(s)$, is associated to each source element s. The specification $as(s)$ determines how accurate is the knowledge on the data satisfying the sources, i.e., how accurate is the source with respect to the associated view $q_\mathcal{G}$. Three possibilities have been considered[1]:

- *Sound views.* When a source s is *sound* (denoted with $as(s) = sound$), its extension provides any subset of the tuples satisfying the corresponding view $q_\mathcal{G}$. In other words, given a source database $\mathcal{D}$, from the fact that a tuple is in $s^\mathcal{D}$ one can conclude that it satisfies the associated view over the global schema, while from the fact that a tuple is not in $s^\mathcal{D}$ one cannot conclude that it does not satisfy the corresponding view. Formally, when $as(s) = sound$, a database $\mathcal{B}$ *satisfies the assertion* $s \leadsto q_\mathcal{G}$ with respect to $\mathcal{D}$ if

$$s^\mathcal{D} \subseteq q_\mathcal{G}^\mathcal{B}$$

  Note that, from a logical point of view, a sound source s with arity n is modeled through the first order assertion

$$\forall \mathbf{x}\ s(\mathbf{x}) \rightarrow q_\mathcal{G}(\mathbf{x})$$

  where $\mathbf{x}$ denotes variables $x_1, \ldots, x_n$.

- *Complete views.* When a source s is *complete* (denoted with $as(s) = complete$), its extension provides any superset of the tuples satisfying the corresponding view. In other words, from the fact that a tuple is in $s^\mathcal{D}$ one cannot conclude that such a tuple satisfies the corresponding view. On the other hand, from the fact that a tuple is not in $s^\mathcal{D}$ one can conclude that such a tuple does not satisfy the view. Formally, when $as(s) = complete$, a database $\mathcal{B}$ *satisfies the assertion* $s \leadsto q_\mathcal{G}$ with respect to $\mathcal{D}$ if

$$s^\mathcal{D} \supseteq q_\mathcal{G}^\mathcal{B}$$

  From a logical point of view, a complete source s with arity n is modeled through the first order assertion

$$\forall \mathbf{x}\ q_\mathcal{G}(\mathbf{x}) \rightarrow s(\mathbf{x})$$

- *Exact Views.* When a source s is *exact* (denoted with $as(s) = exact$), its extension is exactly the set of tuples of objects satisfying the corresponding view. Formally, when $as(s) = exact$, a database $\mathcal{B}$ *satisfies the assertion* $s \leadsto q_\mathcal{G}$ with respect to $\mathcal{D}$ if

$$s^\mathcal{D} = q_\mathcal{G}^\mathcal{B}$$

  From a logical point of view, an exact source s with arity n is modeled through the first order assertion

$$\forall \mathbf{x}\ s(\mathbf{x}) \leftrightarrow q_\mathcal{G}(\mathbf{x})$$

Typically, in the literature, when the specification of $as(s)$ is missing, source s is considered sound. This is also the assumption we make in this paper.

Information Manifold [62], and the system presented in [78] are examples of LAV systems. Information Manifold expresses the global schema in terms of a Description Logic [8], and adopts the language of conjunctive queries as query languages $\mathcal{L}_\mathcal{Q}$, and $\mathcal{L}_{\mathcal{M},\mathcal{G}}$. The system described in [78] uses an XML global schema, and adopts XML-based query languages for both user queries and queries in the mapping. More powerful schema languages for expressing the global schema are reported in [42, 59, 22, 21]. In particular, [42, 59] discusses the case where various forms of relational integrity constraints are expressible in the global schema, including functional and inclusion dependencies, whereas [22, 21] consider a setting where the global schema is expressed in terms of Description Logics [11], which allow for the specification of various types of constraints.

[1]In some papers, for example [24], different assumptions on the domain of the database (open vs. closed) are also taken into account.

3.2 Global as view

In the GAV approach, the mapping $\mathcal{M}$ associates to each element g in $\mathcal{G}$ a query $q_{\mathcal{S}}$ over $\mathcal{S}$. In other words, the query language $\mathcal{L}_{\mathcal{M},\mathcal{G}}$ allows only expressions constituted by one symbol of the alphabet $\mathcal{A}_{\mathcal{G}}$. Therefore, a GAV mapping is a set of assertions, one for each element g of $\mathcal{G}$, of the form

$$g \rightsquigarrow q_{\mathcal{S}}$$

From the modeling point of view, the GAV approach is based on the idea that the content of each element g of the global schema should be characterized in terms of a view $q_{\mathcal{S}}$ over the sources. In some sense, the mapping explicitly tells the system how to retrieve the data when one wants to evaluate the various elements of the global schema. This idea is effective whenever the data integration system is based on a set of sources that is stable. Note that, in principle, the GAV approach favors the system in carrying out query processing, because it tells the system how to use the sources to retrieve data. However, extending the system with a new source is now a problem: the new source may indeed have an impact on the definition of various elements of the global schema, whose associated views need to be redefined.

To better characterize each element of the global schema with respect to the sources, more sophisticated assertions in the GAV mapping can be used, in the same spirit as we saw for LAV. Formally, this means that in the GAV mapping, a new specification, denoted $as(g)$ (either *sound*, *complete*, or *exact*) is associated to each element g of the global schema. When $as(g) = sound$ (resp., *complete*, *exact*), a database $\mathcal{B}$ *satisfies the assertion* $g \rightsquigarrow q_{\mathcal{S}}$ with respect to a source database $\mathcal{D}$ if

$$q_{\mathcal{S}}^{\mathcal{D}} \subseteq g^{\mathcal{B}} \qquad \text{(resp., } q_{\mathcal{S}}^{\mathcal{D}} \supseteq g^{\mathcal{B}},\ q_{\mathcal{S}}^{\mathcal{D}} = g^{\mathcal{B}}\text{)}$$

The logical characterization of sound views and complete views in GAV is therefore through the first order assertions

$$\forall \mathbf{x}\ q_{\mathcal{S}}(\mathbf{x}) \rightarrow g(\mathbf{x}), \qquad \forall \mathbf{x}\ g(\mathbf{x}) \rightarrow q_{\mathcal{S}}(\mathbf{x})$$

respectively.

It is interesting to observe that the implicit assumption in many GAV proposals is the one of exact views. Indeed, in a setting where all the views are exact, there are no constraints in the global schema, and a first order query language is used as $\mathcal{L}_{\mathcal{M},\mathcal{S}}$, a GAV data integration system enjoys what we can call the "single database property", i.e., it is characterized by a single database, namely the global database that is obtained by associating to each element the set of tuples computed by the corresponding view over the sources. This motivates the widely shared intuition that query processing in GAV is easier than in LAV. However, it should be pointed out that the single database property only holds in such a restricted setting.

In particular, the possibility of specifying constraints in $\mathcal{G}$ greatly enhances the modeling power of GAV systems, especially in those situations where the global schema is intended to be expressed in terms of a conceptual data model, or in terms of an ontology [16]. In these cases, the language $\mathcal{L}_{\mathcal{G}}$ is in fact sufficiently powerful to allow for specifying, either implicitly or explicitly, various forms of integrity constraints on the global database.

Most of current data integration systems follow the GAV approach. Notable examples are TSIMMIS [51], Garlic [30], COIN [52], MOMIS [10], Squirrel [92], and IBIS [17]. Analogously to the case of LAV systems, these systems usually adopt simple languages for expressing both the global and the source schemas. IBIS is the only system we are aware of that takes into account integrity constraints in the global schema.

3.3 Comparison between GAV and LAV

The LAV and the GAV approaches are compared in [89] from the point of view of query processing. Generally speaking, it is well known that processing queries in the LAV approach is a difficult task. Indeed, in this approach the only knowledge we have about the data in the global schema is through the views representing the sources, and such views provide only partial information about the data. Since the mapping associates to each source a view over the global schema, it is not immediate to infer how to use the sources in order to answer queries expressed over the global schema. On the other hand, query processing looks easier in the GAV approach, where we can take advantage that the mapping directly specifies which source queries corresponds to the elements of the global schema. Indeed, in most GAV systems, query answering is based on a simple unfolding strategy.

From the point of view of modeling the data integration system, the GAV approach provides a specification mechanism that has a more procedural flavor with respect to the LAV approach. Indeed, while in LAV the designer may concentrate on declaratively specifying the content of the source in terms of the global schema, in GAV, one is forced to specify how to get the data of the global schema by means of queries over the sources. A throughout analysis of the differences/similarities of the two approaches from the point of view of modeling is still missing. A first attempt is reported in [19, 18], where the authors address the problem of checking whether a LAV system can be transformed into a GAV one, and vice-versa. They deal with transformations that are equivalent with respect to query answering, i.e., that enjoy the property that queries posed to the original system have the same answers when posed to the target system. Results on query reducibility from LAV to GAV systems may be useful, for example, to derive a procedural specification from a declarative one. Conversely, results on query reducibility from GAV to LAV may be useful to derive a declarative characterization of the content of the sources starting from a procedural specification. We briefly discuss the notions of query-preserving transformation, and of query-reducibility between classes of data integration systems.

Given two integration systems $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ and $\mathcal{I}' = \langle \mathcal{G}', \mathcal{S}, \mathcal{M}' \rangle$ over the same source schema $\mathcal{S}$ and such that all elements of $\mathcal{G}$ are also elements of $\mathcal{G}'$, $\mathcal{I}'$ is said to be *query-preserving* with respect to $\mathcal{I}$, if for every query q to $\mathcal{I}$ and for every source database $\mathcal{D}$, we have that

$$q^{\mathcal{I},\mathcal{D}} = q^{\mathcal{I}',\mathcal{D}}$$

In other words, $\mathcal{I}'$ is query-preserving with respect to $\mathcal{I}$ if, for each query over the global schema of $\mathcal{I}$ and each source database, the certain answers to the query with respect to the source database that we get from the two integration systems are identical. A class $\mathcal{C}_1$ of integration systems is *query-reducible* to a class $\mathcal{C}_2$ of integration systems if there exist a function $f : \mathcal{C}_1 \rightarrow \mathcal{C}_2$ such that, for each $\mathcal{I}_1 \in \mathcal{C}_1$ we have that $f(\mathcal{I}_1)$ is query-preserving with respect to $\mathcal{I}_1$.

With the two notions in place, the question of query reducibility between LAV and GAV is studied in [18] within a setting where views are considered sound, the global schema is expressed in the relational model, and the queries used in the integration systems (both the queries on the global schema, and the queries in the mapping) are expressed in the language of conjunctive queries. The results show that in such a setting none of the two transformations is possible. On the contrary, if one extends the framework, allowing for integrity constraints in the global schema, then reducibility holds in both directions. In particular, inclusion dependencies and a simple form of equality-generating dependencies suffice for a query-preserving transformation from a LAV system into a GAV one, whereas single head full dependencies are sufficient for the other direction. Both transformations result in a query-preserving system whose size is linearly related to the size of the original one.

Although in this paper we mainly refer to the LAV and GAV approaches to data integration, it is worth noticing that more general types of mappings have been also discussed in the literature. For example, [49] introduces the so-called GLAV approach. In GLAV, the relationships between the global schema and the sources are established by making use of both LAV and GAV assertions. More precisely, in a GLAV mapping as introduced in [49], every assertion has the form $q_\mathcal{S} \leadsto q_\mathcal{G}$, where $q_\mathcal{S}$ is a conjunctive query over the source schema, and $q_\mathcal{G}$ is a conjunctive query over the global schema. A database $\mathcal{B}$ satisfies the assertion $q_\mathcal{S} \leadsto q_\mathcal{G}$ with respect to a source database $\mathcal{D}$ if $q_\mathcal{S}^\mathcal{D} \subseteq q_\mathcal{G}^\mathcal{B}$. Thus, the GLAV approach models a situation where sources are sound. Interestingly, the technique presented in [19, 18] can be extended for transforming any GLAV system into a GAV one. The key idea is that a GLAV assertion can be transformed into a GAV assertion plus an inclusion dependency. Indeed, for each assertion

$$q_\mathcal{S} \leadsto q_\mathcal{G}$$

in the GLAV system (where the arity of both queries is n), we introduce a new relation symbol r of arity n in the global schema of the resulting GAV system, and we associate to r the sound view $q_\mathcal{S}$ by means of

$$r \leadsto q_\mathcal{S}$$

plus the inclusion dependency

$$r \subseteq q_\mathcal{G}.$$

Now, it is immediate to verify that the above inclusion dependency can be treated exactly with the same technique introduced in the LAV to GAV transformation, and therefore, from the GLAV system we can obtain a query-preserving GAV system whose size is linearly related to the size of the original system.

4. QUERY PROCESSING IN LAV

In this section we discuss query processing in the LAV approach. From the definition given in Section 3, it is easy to see that answering queries in LAV systems is essentially an extended form of reasoning in the presence of incomplete information [91]. Indeed, when we answer a query over the global schema on the basis of a LAV mapping, we know only the extensions of the views associated to the sources, and this provides us with only partial information on the global database. As we already observed, in general, there are several possible global databases that are legal for the data integration system with respect to a given source database. This observation holds even for a setting where only sound views are allowed in the mapping. The problem is even more complicated when sources can be modeled as complete or exact views. In particular, dealing with exact sources essentially means applying the closed world assumption on the corresponding views [1, 85].

The following example rephrases an example given in [1]. Consider a data integration system $\mathcal{I}$ with global relational schema $\mathcal{G}$ containing (among other relations) a binary relation couple, and two constants Ann and Bill. Consider also two sources female and male, respectively with associated views

$$\begin{aligned}\mathsf{female}(f) &\leadsto \{\ f, m \mid \mathsf{couple}(f, m)\ \}\\ \mathsf{male}(m) &\leadsto \{\ f, m \mid \mathsf{couple}(f, m)\ \}\end{aligned}$$

and consider a source database $\mathcal{D}$ with $\mathsf{female}^\mathcal{D} = \{\mathsf{Ann}\}$ and $\mathsf{male}^\mathcal{D} = \{\mathsf{Bill}\}$, and assume that there are no constraints imposed by a schema. If both sources are sound, we only know that some couple has Ann as its female component and Bill as its male component. Therefore, the query

$$Q \ = \ \{\ x, y \mid \mathsf{couple}(x, y)\ \}$$

asking for all couples would return an empty answer, i.e., $Q_c^{\mathcal{I},\mathcal{D}} = \emptyset$. However, if both sources are exact, we can conclude that all couples have Ann as their female component and Bill as their male component, and hence that $(\mathsf{Ann}, \mathsf{Bill})$ is the only couple, i.e., $Q_c^{\mathcal{I},\mathcal{D}} = \{(\mathsf{Ann}, \mathsf{Bill})\}$.

Since in LAV, sources are modeled as views over the global schema, the problem of processing a query is traditionally called *view-based query processing*. Generally speaking, the problem is to compute the answer to a query based on a set of views, rather than on the raw data in the database [89, 60].

There are two approaches to view-based query processing, called *view-based query rewriting* and *view-based query answering*, respectively. In the former approach, we are given a query q and a set of view definitions, and the goal is to reformulate the query into an expression of a fixed language $\mathcal{L}_R$ that refers only to the views and provides the answer to q. The crucial point is that the language in which we want the rewriting is fixed, and in general coincides with the language used for expressing the original query. In a LAV data integration setting, query rewriting aims at reformulating, in a way that is independent from the current source database, the original query in terms of a query to the sources. Obviously, it may happen that no rewriting in the target language $\mathcal{L}_R$ exists that is equivalent to the original query. In this case, we are interested in computing a so-called *maximally contained rewriting*, i.e., an expression that captures the original query in the best way.

Sound	CQ	$CQ^{\neq}$	PQ	Datalog	FOL
CQ	*PTIME*	*coNP*	*PTIME*	*PTIME*	*undec.*
$CQ^{\neq}$	*PTIME*	*coNP*	*PTIME*	*PTIME*	*undec.*
PQ	*coNP*	*coNP*	*coNP*	*coNP*	*undec.*
Datalog	*coNP*	*undec.*	*coNP*	*undec.*	*undec.*
FOL	*undec.*	*undec.*	*undec.*	*undec.*	*undec.*
Exact	CQ	$CQ^{\neq}$	PQ	Datalog	FOL
CQ	*coNP*	*coNP*	*coNP*	*coNP*	*undec.*
$CQ^{\neq}$	*coNP*	*coNP*	*coNP*	*coNP*	*undec.*
PQ	*coNP*	*coNP*	*coNP*	*coNP*	*undec.*
Datalog	*undec.*	*undec.*	*undec.*	*undec.*	*undec.*
FOL	*undec.*	*undec.*	*undec.*	*undec.*	*undec.*

Table 1: Complexity of view-based query answering

In view-based query answering, besides the query q and the view definitions, we are also given the extensions of the views. The goal is to compute the set of tuples t such that the knowledge on the view extensions logically implies that t is an answer to q, i.e., t is in the answer to q in all the databases that are consistent with the views. It is easy to see that, in a LAV data integration framework, this is exactly the problem of computing the certain answers to q with respect to a source database.

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

A large number of results have been reported for both approaches. We first focus on view-based query answering.

Query answering has been extensively investigated in the last years [1, 53, 43, 66, 4, 21]. A comprehensive framework for view-based query answering, as well as several interesting results, is presented in [53]. The framework considers various assumptions for interpreting the view extensions with respect to the corresponding definitions (closed, open, and exact view assumptions). In [1], an analysis of the complexity of the problem under the different assumptions is carried out for the case where the views and the queries are expressed in terms of various languages (conjunctive queries without and with inequalitites, positive queries, Datalog, and first-order queries). The complexity is measured with respect to the size of the view extensions (data complexity). Table 1 summarizes the results presented in [1]. Note that, for the query languages considered in that paper, the exact view assumption complicates the problem. For example, the data complexity of query answering for the case of conjunctive queries is PTIME under the sound view assumption, and coNP-complete for exact views. This can be explained by noticing that the exact view assumption introduces a form of negation, and therefore it may force to reason by cases on the objects stored in the views.

In [24], the problem is studied for a setting where the global schema models a semistructured database, i.e., a labeled directed graphs. It follows that both the user queries, and the queries used in the LAV mapping should be expressed in a query language for semistructured data. The main difficulty arising in this context is that languages for querying semistructured data enable expressing regular-path queries [2, 15, 45]. A regular-path query asks for all pairs of nodes in the database connected by a path conforming to a regular expression, and therefore may contain a restricted form of recursion. Note that, when the query contains unrestricted recursion, both view-based query rewriting and view-based query answering become undecidable, even when the views are not recursive [43].

Table 2 summarizes the results presented in [24]. Both data complexity, and expression complexity (complexity with respect to the size of the query and the view definitions) are taken into account. All upper bound results have been obtained by automata-theoretic techniques. In the analysis, a further distinction is proposed for characterizing the domain of the database (open vs. closed domain assumption). In the closed domain assumption we assume that the global database contains only objects stored in the sources. The results show that none of the cases can be solved in polynomial time (unless P = NP). This can be explained by observing that the need for considering various forms of incompleteness expressible in the query language (due to union and transitive closure), is a source of complexity for query answering. Obviously, under closed domain, our knowledge is more accurate than in the case of the open domain assumption, and this rules out the need for some combinatorial reasoning. This provides the intuition of why under closed domain the problem is "only" coNP-complete in all cases, for data, expression, and combined complexity. On the other hand, under open domain, we cannot exclude the possibility that the database contains more objects than those known to be in the views. For combined complexity, this means that we are forced to reason about the definition of the query and the views. Indeed, the problem cannot be less complex than comparing two regular path queries, and this explains the PSPACE lower bound. Interestingly, the table shows that the problem does not exceed the PSPACE complexity. Moreover, the data complexity remains in coNP, and therefore, although we are using a query language that is powerful enough to express a (limited) form of recursion, the problem is no more complex than in the case of disjunctions of conjunctive queries [1].

While regular-path queries represent the core of any query language for semistructured data, their expressive power is limited. Several authors point out that extensions are required for making them useful in real settings (see for example [14, 15, 80]). Indeed, the results in [24] have been extended to query language with the inverse operator [26], and to the class of union of conjunctive regular-path queries in [28].

Turning our attention to view-based query rewriting, several recent papers investigate the rewriting question for different classes of queries. The problem is investigated for the case of conjunctive queries (with or without arithmetic comparisons) in [66, 84], for disjunctive views in [4], for queries with aggregates in [87, 37, 56], for recursive queries and nonrecursive views in [43], for queries expressed in Description Logics in [9], for regular-path queries and their extensions

domain	views	Complexity data	Complexity expression	Complexity combined
closed	all sound	*coNP*	*coNP*	*coNP*
	all exact	*coNP*	*coNP*	*coNP*
	arbitrary	*coNP*	*coNP*	*coNP*
open	all sound	*coNP*	*PSPACE*	*PSPACE*
	all exact	*coNP*	*PSPACE*	*PSPACE*
	arbitrary	*coNP*	*PSPACE*	*PSPACE*

Table 2: Complexity of view-based query answering for regular-path queries

in [23, 26, 27], and in the presence of integrity constraints in [59, 44]. Rewriting techniques for query optimization are described, for example, in [34, 3, 88], and in [46, 80, 82] for the case of path queries in semistructured data.

We already noted that view-based query rewriting and view-based query answering are different problems. Unfortunately, their similarity sometimes gives 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, the best possible rewriting is expressible as union of conjunctive queries, which is basically the same language as the one of the original query and views. However, for other query languages this is not the case. Abstracting from the language used to express the rewriting, we can 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 with respect to 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. i.e., in data integration terminology, the set of certain answers to the query with respect to the view extension. Hence, the perfect rewriting is the best rewriting 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 [1]. However, 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. By establishing a tight connection between view-based query answering and constraint-satisfaction problems, it is argued in [27] that this is a difficult task.

5. QUERY PROCESSING IN GAV

Most GAV data integration systems do not allow integrity constraints in the global schema, and assume that views are exact. It is easy to see that, under these assumptions, query processing can be based on a simple unfolding strategy. When we have a query q over the alphabet $\mathcal{A}_\mathcal{G}$ of the global schema, every element of $\mathcal{A}_\mathcal{G}$ is substituted with the corresponding query over the sources, and the resulting query is then evaluated at the sources. As we said before, such a strategy suffices mainly because the data integration system enjoys the single database property. Notably, the same strategy applies also in the case of sound views.

However, when the language $\mathcal{L}_\mathcal{G}$ used for expressing the global schema allows for integrity constraints, and the views are sound, then query processing in GAV systems becomes more complex. Indeed, in this case, integrity constraints can in principle be used in order to overcome incompleteness of data at the sources. The following example shows that, by taking into account foreign key constraints, one can obtain answers that would be missed by simply unfolding the user query.

$s_1^\mathcal{D}$:

12	*calvin*	*rome*	21
15	*alice*	*hong kong*	24

$s_2^\mathcal{D}$:

AF	*hotdog corp.*
BN	*banana ltd.*

$s_3^\mathcal{D}$:

12	*AF*
16	*BN*

Figure 1: Extension of sources for the example

Let $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a data integration system, where $\mathcal{G}$ is constituted by the relations

$$\begin{array}{l} \mathsf{employee}(Ecode, Ename, Ecity) \\ \mathsf{company}(Ccode, Cname) \\ \mathsf{employed}(Ecode, Ccode) \end{array}$$

and the constraints

$$\begin{array}{rcl} key(\mathsf{employee}) & = & \{Ecode\} \\ key(\mathsf{company}) & = & \{Ccode\} \\ \mathsf{employed}[Ecode] & \subseteq & \mathsf{employee}[Ecode] \\ \mathsf{employed}[Ccode] & \subseteq & \mathsf{company}[Ccode] \end{array}$$

The source schema $\mathcal{S}$ consists of three sources. Source s_1, of arity 4, contains information about employees with their code, name, city, and date of birth. Source s_2, of arity 2, contains codes and names of companies. Finally, Source s_3, of arity 2, contains information about employment in companies. The mapping $\mathcal{M}$ is defined by

$$\begin{array}{rl} \mathsf{employee} \rightsquigarrow & \{\ x, y, z \mid \mathsf{s}_1(x, y, z, w)\ \} \\ \mathsf{company} \rightsquigarrow: & \{\ x, y \mid \mathsf{s}_2(x, y)\ \} \\ \mathsf{employed} \rightsquigarrow: & \{\ x, w \mid \mathsf{s}_3(x, w)\ \} \end{array}$$

Now consider the following user query q, asking for codes of employees:

$$\{\ x \mid \mathsf{employee}(x, y, z)\ \}$$

Suppose that the data stored in the source database $\mathcal{D}$ are those depicted in Figure 1: by simply unfolding q we obtain the answer $\{12\}$. However, due to the integrity constraint $\mathsf{employed}[Ecode] \subseteq \mathsf{employee}[Ecode]$, we know that 16 is the code of a person, even if it does not appear in $s_1^\mathcal{D}$. The correct answer to q is therefore $\{12, 16\}$. Observe that we do not know any value for the attributes of the employee whose *Ecode* is 16.

Given a source database $\mathcal{D}$, let us call "retrieved global database" the global database that is obtained by populating each relation r in the global schema according to the mapping, i.e., by populating r with the tuples obtained by evaluating the query that the mapping associates to q. In general, integrity constraints may be violated in the retrieved global database (e.g., the retrieved global database for the above example). Regarding key constraints, let us assume that the query that the mapping associates to each global schema relation r is such that the data retrieved for r do not violate the key constraint of r. In other words, the management of key constraints is left to the designer (see next section for a discussion on this subject). On the other hand, the management of foreign key constraints cannot be left to the designer, since it is strongly related to the incompleteness of the sources. Moreover, since foreign keys are interrelation constraints, they cannot be dealt with in the GAV mapping, which, by definition, works on each global relation in isolation.

The assumption of sound views asserts that the tuples retrieved for a relation r are a subset of the tuples that the system assigns to r; therefore, we may think of completing the retrieved global database by suitably adding tuples in order to satisfy foreign key constraints, while still conforming to the mapping. When a foreign key constraint is violated, there are several ways of adding tuples to the retrieved global database to satisfy such a constraint. In other words, in the presence of foreign key constraints in the global schema, the semantics of a data integration system must be formulated in terms of a *set* of databases, instead of a single one. Since we are interested in the certain answers $q^{\mathcal{I},\mathcal{D}}$ to a query q, i.e., the tuples that satisfy q in all global databases that are legal for $\mathcal{I}$ with respect to $\mathcal{D}$, the existence of several such databases complicates the task of query answering.

In [17], a system called IBIS is presented, that takes into account key and foreign key constraints over the global relational schema. The system uses the foreign key constraints in order to retrieve data that could not be obtained in traditional data integration systems. The language for expressing both the user query and the queries in the mapping is the one of union of conjunctive queries. To process a query q, IBIS expands q by taking into account the foreign key constraints on the global relations appearing in the atoms. Such an expansion is performed by viewing each foreign key constraint $r_1[\mathbf{X}] \subseteq r_2[\mathbf{Y}]$, where $\mathbf{X}$ and $\mathbf{Y}$ are sets of h attributes and $\mathbf{Y}$ is a key for r_2, as a logic programming [77] rule

$$r'_2(\vec{X}, f_{h+1}(\vec{X}), \ldots, f_n(\vec{X})) \leftarrow r'_1(\vec{X}, X_{h+1}, \ldots, X_m)$$

where each f_i is a Skolem function, $\vec{X}$ is a vector of h variables, and we have assumed for simplicity that the attributes involved in the foreign key are the first h ones. Each r'_i is a predicate, corresponding to the global relation r_i, defined by the above rules for foreign key constraints, together with the rule

$$r'_i(X_1, \ldots, X_n) \leftarrow r_i(X_1, \ldots, X_n)$$

Once such a logic program $\Pi_{\mathcal{G}}$ has been defined, it can be used to generate the expanded query *expand_q* associated to the original query q. This is done by performing a partial evaluation [40] with respect to $\Pi_{\mathcal{G}}$ of the body of q', which is the query obtained by substituting in q each predicate r_i with r'_i. In the partial evaluation tree, a node is not expanded anymore either when no atom in the node unifies with a head of a rule, or when the node is subsumed by (i.e., is more specific than) one of its predecessors. In the latter case, the node gets an empty node as a child; intuitively this is because such a node cannot provide any answer that is not already provided by its more general predecessor. These conditions guarantee that the construction of the partial evaluation tree for a query always terminates. Then, the expansion *expand_q* of q is a union of conjunctive queries whose body is constituted by the disjunction of all nonempty leaves of the partial evaluation tree. It is possible to show that, by unfolding *expand_q* according to the mapping, and evaluating the resulting query over the sources, one obtains exactly the set of certain answers of q to $\mathcal{I}$ with respect to $\mathcal{D}$ [17].

6. INCONSISTENCIES BETWEEN SOURCES

The formalization of data integration presented in the previous sections is based on a first order logic interpretation of the assertions in the mapping, and, therefore, is not able to cope with inconsistencies between sources. Indeed, if in a data integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, the data retrieved from the sources do not satisfy the integrity constraints of $\mathcal{G}$, then no global database exists for $\mathcal{I}$, and query answering becomes meaningless. This is the situation occurring when data in the sources are mutually inconsistent. In practice, this situation is generally dealt with by means of suitable transformation and cleaning procedures to be applied to data retrieved by the sources (see [12, 50]). In this section, we address the problem from a more theoretical perspective.

Several recent papers aim at formally dealing with inconsistencies in databases, in particular for providing informative answers even in the case of a database that does not satisfy its integrity constraints (see, for example, [13, 6, 7, 54]). Although interesting, such results are not specifically tailored to the case of different consistent data sources that are mutually inconsistent, that is the case of interest in data integration. This case is addressed in [76], where the authors propose an operator for *merging databases* under constraints. Such operator allows one to obtain maximal amount of information from each database by means of a majority criterion used in case of conflict. However, also the approach described in [76] does not take explicitly into account the notion of mapping as introduced in our data integration setting.

In data integration, according to the definition of mapping satisfaction as given in Section 3, it may be the case that the data retrieved from the sources cannot be reconciled in the global schema in such a way that both the constraints of the global schema, and the mapping are satisfied. For example, this happens when a key constraint specified for the relation r in the global schema is violated by the tuples retrieved by the view associated to r, since the assumption of sound views does not allow us to disregard tuples from

r with duplicate keys. If we do not want to conclude in this case that no global database exists that is legal for $\mathcal{I}$ with respect to $\mathcal{D}$, we need a different characterization of the mapping. In particular, we need a characterization that allows us support query processing even when the data at the sources are incoherent with respect to the integrity constraints on the global schema.

A possible solution is to characterize the data integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ (with $\mathcal{M} = \{r_1 \leadsto V_1, \ldots, r_n \leadsto V_n\}$), in terms of those global databases that

1. satisfy the integrity constraints of $\mathcal{G}$, and
2. approximate at best the satisfaction of the assertions in the mapping $\mathcal{M}$, i.e., that are *as sound as possible*.

In other, the integrity constraints of $\mathcal{G}$ are considered strong, whereas the mapping is considered soft. Given a source database $\mathcal{D}$ for $\mathcal{I}$, we can now define an ordering between the global databases for $\mathcal{I}$ as follows. If $\mathcal{B}_1$ and $\mathcal{B}_2$ are two databases that are legal with respect to $\mathcal{G}$, we say that $\mathcal{B}_1$ is *better* than $\mathcal{B}_2$ with respect to $\mathcal{D}$, denoted as $\mathcal{B}_1 \gg_{\mathcal{D}} \mathcal{B}_2$, if there exists an assertion $r_i \leadsto V_i$ in $\mathcal{M}$ such that

- $(r_i^{\mathcal{B}_1} \cap V_i^{\mathcal{D}}) \supset (r_i^{\mathcal{B}_2} \cap V_i^{\mathcal{D}})$, and
- $(r_j^{\mathcal{B}_1} \cap V_j^{\mathcal{D}}) \supseteq (r_j^{\mathcal{B}_2} \cap V_j^{\mathcal{D}})$, for all $r_j \leadsto V_j$ in $\mathcal{M}$ with $j \neq i$;

Intuitively, this means that there is at least one assertion for which $\mathcal{B}_1$ satisfies the sound mapping better than $\mathcal{B}_2$, while for no other assertion $\mathcal{B}_2$ is better than $\mathcal{B}_1$. In other words, $\mathcal{B}_1$ approximates the sound mapping better than $\mathcal{B}_2$.

It is easy to verify that the relation $\gg_{\mathcal{D}}$ is a partial order. With this notion in place, we can now define the notion of $\mathcal{B}$ satisfying the mapping $\mathcal{M}$ with respect to $\mathcal{D}$ in our setting: a database $\mathcal{B}$ that is legal with respect to $\mathcal{G}$ satisfies the mapping $\mathcal{M}$ with respect to $\mathcal{D}$ if $\mathcal{B}$ is maximal with respect to $\gg_{\mathcal{D}}$, i.e., for no other global database $\mathcal{B}'$ that is legal with respect to $\mathcal{G}$, we have that $\mathcal{B}' \gg_{\mathcal{D}} \mathcal{B}$.

The notion of legal database for $\mathcal{I}$ with respect to $\mathcal{D}$, and the notion of certain answer remain the same, given the new definition of satisfaction of mapping. It is immediate to verify that, if there exists a legal database for $\mathcal{I}$ with respect to $\mathcal{D}$ under the first order logic interpretation of the mapping, then the new semantics and the old one coincide, in the sense that, for each query q, the set $q^{\mathcal{I},\mathcal{D}}$ of certain answers computed under the first order semantics coincides with the set of certain answers computed under the new semantics presented here.

The problem of inconsistent sources in data integration is addressed in [64], in particular for the case where:

- the global schema is a relational schema with key and foreign key constraints,
- the mapping is of type GAV,
- the query language $\mathcal{L}_{\mathcal{M},\mathcal{S}}$ is the language of union of conjunctive queries,
- the views in the mapping are intended to be sound.

In such a setting, an algorithm is proposed for computing the certain answers of a query in the new semantical framework presented above. The algorithm checks whether a given tuple t is a certain answer to a query q with respect to a given source database $\mathcal{D}$ in coNP data complexity (i.e., with respect to the size of $\mathcal{D}$). Based on this result, the problem of computing the certain answers in the presented framework can be shown to be coNP-complete in data complexity.

7. REASONING ON QUERIES

Recent work addresses the problem of reasoning on queries in data integration systems. The basic form of reasoning on queries is checking containment, i.e., verifying whether one query returns a subset of the result computed by the other query in all databases. Most of the results on query containment concern conjunctive queries and their extensions. In [33], NP-completeness has been established for conjunctive queries, in [63, 90], Π_2^p-completeness of containment of conjunctive queries with inequalities is proved, and in [86] the case of queries with the union and difference operators is studied. For various classes of Datalog queries with inequalities, decidability and undecidability results are presented in [35] and [90], respectively. Other papers consider the case of query containment in the presence of various types of constraints [5, 39, 32, 69, 71, 70, 20], and for regular-path queries and their extensions [47, 25, 28, 41].

Besides the usual notion of containment, several other notions have been introduced related to the idea of comparing queries in a data integration setting, especially in the context of the LAV approach.

In [79], a query is said to be contained in another query *relative to a set of sources* modeled as views, if, for each extension of the views, the certain answers to the former query are a subset of the certain answers to the latter. Note that this reasoning problem is different from the usual containment checking: here we are comparing the two queries with respect to the certain answers computable on the basis of the views available. The difference becomes evident if one considers a counterexample to relative containment: Q_1 is not contained in Q_2 relative to views $\mathcal{V}$ if there is a tuple t and an extension $\mathcal{E}$ of $\mathcal{V}$, such that for each database $\mathcal{DB}$ consistent with $\mathcal{E}$ (i.e., a database $\mathcal{DB}$ such that, the result $\mathcal{V}^{\mathcal{DB}}$ of evaluating the views over $\mathcal{DB}$ is exactly $\mathcal{E}$), t is an answer of Q_1 to $\mathcal{DB}$, but there is a database $\mathcal{DB}'$ consistent with $\mathcal{E}$ such that t is not an answer of Q_2 to $\mathcal{DB}'$. In other words, Q_1 is not contained in Q_2 relative to views $\mathcal{V}$ if there are two databases $\mathcal{DB}$ and $\mathcal{DB}'$ such that $\mathcal{V}^{\mathcal{DB}} = \mathcal{V}^{\mathcal{DB}'}$ and $Q_1^{\mathcal{DB}} = Q_2^{\mathcal{DB}'}$.

In [79], it is shown that the problem of checking relative containment is Π_2^P complete in the case of conjunctive queries and views. In [74], such results are extended to the case where views have limited access patterns.

In [72], the authors introduce the notion of "p-containment" (where "p" stands for power): a view set $\mathcal{V}$ is said to be p-contained in another view set $\mathcal{W}$, i.e., $\mathcal{W}$ has at least the answering power of $\mathcal{V}$, if $\mathcal{W}$ can answer all queries that can be answered using $\mathcal{V}$.

The notion of "information content" of materialized views is studied in [57] for a restricted class of aggregate queries, with the goal of devising techniques for checking whether a set of views is sufficient for completely answering a given query based on the views.

One of the ideas underlying the above mentioned papers is the one of losslessness: a set of views is *lossless* with respect to a query, if, no matter what the database is, we can answer the query by solely relying on the content of the views. This question is relevant for example in mobile computing, where we may be interested in checking whether a set of cached data allows us to derive the requested information without accessing the network, or in data warehouse design, in particular for the view selection problem [36], where we have to measure the quality of the choice of the views to materialize in the data warehouse. In data integration, losslessness may help in the design of the data integration system, in particular, by selecting a minimal subset of sources to access without losing query-answering power.

The definition of losslessness relies on that of certain answers: a set of views is *lossless* with respect to a query, if for every database, we can answer the query over that database by computing the certain answers based on the view extensions. It follows that there are at least two versions of losslessness, namely, losslessness under the sound view assumption, and losslessness under the exact view assumption.

The first version is obviously weaker than the second one. If views $\mathcal{V}$ are lossless with respect to a query Q under the sound view assumption, then we know that, from the intensional point of views, $\mathcal{V}$ contain enough information to completely answer Q, even though the possible incompleteness of the view extensions may prevent us form obtaining all the answers that Q would get from the database. On the other hand, if $\mathcal{V}$ are lossless with respect to a query Q under the exact view assumption, then we know that they contain enough information to completely answer Q, both from the intensional and from the extensional point of view.

In [29], the problem of losslessness is addressed in a context where both the query and the views are expressed as regular path queries. It is shown that, in the case of the sound view assumption, the problem is solvable by a technique that is based on searching for a counterexample to losslessness, i.e., two databases that are both coherent with the view extensions, and that differ in the answers to the query. Different from traditional query containment, the search for a counterexample is complicated by the presence of a quantification over all possible view extensions. The key observation in [29] is that, under the sound view assumption, we can restrict our attention to counterexamples that are linear databases, and this allows devising a method that uses, via automata-theoretic techniques, the known connection between view-based query answering and constraint satisfaction [27]. As far as the computational complexity is concerned, the problem is PSPACE-complete with respect to the view definitions, and EXPSPACE-complete with respect to the query.

It is interesting to observe that, for the case of exact views, the search for a counterexample cannot be restricted to linear databases. Actually, the question of losslessness under the exact view assumption is largely unexplored. To the best of our knowledge, the problem is open even for a setting where both the query and the views are conjunctive queries.

8. CONCLUSIONS

The aim of this tutorial was to provide an overview of some of the theoretical issues underlying data integration. Several interesting problems remain open in each of the topics that we have discussed. For example, more investigation is needed for a deep understanding of the relationship between the LAV and the GAV approaches. Open problems remain on algorithms and complexity for view-based query processing, in particular for the case of rich languages for semistructured data, for the case of exact views, and for the case of integrity constraints in the global schema. Query processing in GAV with constraints has been investigated only recently, and interesting classes of constraints have not been considered yet. The treatment of mutually inconsistent sources, and the issue of reasoning on queries present many open research questions.

Moreover, data integration is such a rich field that several important related aspects not addressed here can be identified, including the following.

- How to build an appropriate global schema, and how to discover inter-schema [31] and mapping assertions (LAV or GAV) in the design of a data integration system (see, for instance, [83]).
- How to (automatically) synthesize wrappers that present the data at the sources in a form [] that is suitable for their use in the mapping.
- How to deal with possible limitations in accessing the sources, both in LAV [84, 67, 68] and in GAV [75, 48, 73, 74].
- How to incorporate the notions of quality (data quality, quality of answers, etc.) [81], and data cleaning [12] into a formal framework for data integration.
- How to learn rules that allow for automatically mapping data items in different sources (for example, for inferring that two key values in different sources actually refer to the same real-world object [38]).
- How to go beyond the architecture based on a global schema, so as, for instance, to model data exchange, transformation, and cooperation rather than data integration (see, e.g., [55]), or to devise information integration facilities for the Semantic Web.
- How to optimize the evaluation of queries posed to a data integration system [3].

We believe that each of the above issues is characterized by interesting research problems still to investigate.

9. ACKNOWLEDGMENTS

I warmly thank Diego Calvanese, Giuseppe De Giacomo and Moshe Y. Vardi, with whom I carried out most of my research work on data integration during the last years. Also, I thank all colleagues that I have been working with in several data integration projects, in particular Andrea Calì, Domenico Lembo, Daniele Nardi, Riccardo Rosati, and all participants to the ESPRIT LTR project "DWQ (Data Warehouse Quality)", and the MIUR (Italian Ministry of University and Research) project "D2I (From Data To Information)".

Finally, this is the first paper I can dedicate to my son Domenico, and for that I want to thank his mother.

10. REFERENCES

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

[2] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. L. Wiener. The Lorel query language for semistructured data. *Int. J. on Digital Libraries*, 1(1):68–88, 1997.

[3] S. Adali, K. S. Candan, Y. Papakonstantinou, and V. S. Subrahmanian. Query caching and optimization in distributed mediator systems. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 137–148, 1996.

[4] F. N. Afrati, M. Gergatsoulis, and T. Kavalieros. Answering queries using materialized views with disjunction. In *Proc. of the 7th Int. Conf. on Database Theory (ICDT'99)*, volume 1540 of *Lecture Notes in Computer Science*, pages 435–452. Springer, 1999.

[5] A. V. Aho, Y. Sagiv, and J. D. Ullman. Equivalence among relational expressions. *SIAM J. on Computing*, 8:218–246, 1979.

[6] M. Arenas, L. E. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *Proc. of the 18th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'99)*, pages 68–79, 1999.

[7] M. Arenas, L. E. Bertossi, and J. Chomicki. Specifying and querying database repairs using logic programs with exceptions. In *Proc. of the 4th Int. Conf. on Flexible Query Answering Systems (FQAS'00)*, pages 27–41. Springer, 2000.

[8] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications.* Cambridge University Press, 2002. To appear.

[9] C. Beeri, A. Y. Levy, and M.-C. Rousset. Rewriting queries using views in description logics. In *Proc. of the 16th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'97)*, pages 99–108, 1997.

[10] D. Beneventano, S. Bergamaschi, S. Castano, A. Corni, R. Guidetti, G. Malvezzi, M. Melchiori, and M. Vincini. Information integration: the MOMIS project demonstration. In *Proc. of the 26th Int. Conf. on Very Large Data Bases (VLDB 2000)*, 2000.

[11] A. Borgida. Description logics in data management. *IEEE Trans. on Knowledge and Data Engineering*, 7(5):671–682, 1995.

[12] M. Bouzeghoub and M. Lenzerini. Introduction to the special issue on data extraction, cleaning, and reconciliation. *Information Systems*, 26(8):535–536, 2001.

[13] F. Bry. Query answering in information systems with integrity constraints. In *IFIP WG 11.5 Working Conf. on Integrity and Control in Information System.* Chapman & Hall, 1997.

[14] P. Buneman. Semistructured data. In *Proc. of the 16th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'97)*, pages 117–121, 1997.

[15] P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu. A query language and optimization technique for unstructured data. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 505–516, 1996.

[16] A. Calì, D. Calvanese, G. De Giacomo, and M. Lenzerini. Accessing data integration systems through conceptual schemas. In *Proc. of the 20th Int. Conf. on Conceptual Modeling (ER 2001)*, 2001.

[17] A. Calì, D. Calvanese, G. De Giacomo, and M. Lenzerini. Data integration under integrity constraints. In *Proc. of the 14th Conf. on Advanced Information Systems Engineering (CAiSE 2002)*, 2002. To appear.

[18] A. Calì, D. Calvanese, G. De Giacomo, and M. Lenzerini. On the expressive power of data integration systems. Submitted for pubblication, 2002.

[19] A. Calì, G. De Giacomo, and M. Lenzerini. Models of information integration: Turning local-as-view into global-as-view. In *Foundations of Models for Information Integration.* On line proceedings, `http://www.fmldo.org/FMII-2001`, 2001.

[20] D. Calvanese, G. De Giacomo, and M. Lenzerini. On the decidability of query containment under constraints. In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'98)*, pages 149–158, 1998.

[21] D. Calvanese, G. De Giacomo, and M. Lenzerini. Answering queries using views over description logics knowledge bases. In *Proc. of the 17th Nat. Conf. on Artificial Intelligence (AAAI 2000)*, pages 386–391, 2000.

[22] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, and R. Rosati. Description logic framework for information integration. In *Proc. of the 6th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 2–13, 1998.

[23] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. Rewriting of regular expressions and regular path queries. In *Proc. of the 18th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'99)*, pages 194–204, 1999.

[24] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. 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.

[25] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. Containment of conjunctive regular path queries with inverse. In *Proc. of the 7th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2000)*, pages 176–185, 2000.

[26] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. Query processing using views for regular path queries with inverse. In *Proc. of the 19th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2000)*, pages 58–66, 2000.

[27] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. View-based query processing and constraint satisfaction. In *Proc. of the 15th IEEE Symp. on Logic in Computer Science (LICS 2000)*, pages 361–371, 2000.

[28] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. View-based query answering and query containment over semistructured data. In *Proc. of the 8th Int. Workshop on Database Programming Languages (DBPL 2001)*, 2001.

[29] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. Lossless regular views. In *Proc. of the 21st ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2002)*, pages 58–66, 2002.

[30] M. J. Carey, L. M. Haas, P. M. Schwarz, M. Arya, W. F. Cody, R. Fagin, M. Flickner, A. Luniewski, W. Niblack, D. Petkovic, J. Thomas, J. H. Williams, and E. L. Wimmers. Towards heterogeneous multimedia information systems: The Garlic approach. In *Proc. of the 5th Int. Workshop on Research Issues in Data Engineering – Distributed Object Management (RIDE-DOM'95)*, pages 124–131. IEEE Computer Society Press, 1995.

[31] T. Catarci and M. Lenzerini. Representing and using interschema knowledge in cooperative information systems. *J. of Intelligent and Cooperative Information Systems*, 2(4):375–398, 1993.

[32] E. P. F. Chan. Containment and minimization of positive conjunctive queries in oodb's. In *Proc. of the 11th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'92)*, pages 202–211, 1992.

[33] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proc. of the 9th ACM Symp. on Theory of Computing (STOC'77)*, pages 77–90, 1977.

[34] S. Chaudhuri, S. Krishnamurthy, S. Potarnianos, and K. Shim. Optimizing queries with materialized views. In *Proc. of the 11th IEEE Int. Conf. on Data Engineering (ICDE'95)*, Taipei (Taiwan), 1995.

[35] S. Chaudhuri and M. Y. Vardi. On the equivalence of recursive and nonrecursive Datalog programs. In *Proc. of the 11th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'92)*, pages 55–66, 1992.

[36] R. Chirkova, A. Y. Halevy, and D. Suciu. A formal perspective on the view selection problem. In *Proc. of the 27th Int. Conf. on Very Large Data Bases (VLDB 2001)*, pages 59–68, 2001.

[37] S. Cohen, W. Nutt, and A. Serebrenik. Rewriting aggregate queries using views. In *Proc. of the 18th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'99)*, pages 155–166, 1999.

[38] W. W. Cohen. Integration of heterogeneous databases without common domains using queries based on textual similarity. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 201–212, 1998.

[39] A. C. K. David S. Johnson. Testing containment of conjunctive queries under functional and inclusion dependencies. *J. of Computer and System Sciences*, 28(1):167–189, 1984.

[40] G. De Giacomo. Intensional query answering by partial evaluation. *J. of Intelligent Information Systems*, 7(3):205–233, 1996.

[41] A. Deutsch and V. Tannen. Optimization properties for classes of conjunctive regular path queries. In *Proc. of the 8th Int. Workshop on Database Programming Languages (DBPL 2001)*, 2001.

[42] O. Duschka. *Query Planning and Optimization in Information Integration.* PhD thesis, Stanford University, 1997.

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

[44] O. M. Duschka and A. Y. Levy. Recursive plans for information gathering. In *Proc. of the 15th Int. Joint Conf. on Artificial Intelligence (IJCAI'97)*, pages 778–784, 1997.

[45] M. F. Fernandez, D. Florescu, J. Kang, A. Y. Levy, and D. Suciu. Catching the boat with strudel: Experiences with a web-site management system. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 414–425, 1998.

[46] M. F. Fernandez and D. Suciu. Optimizing regular path expressions using graph schemas. In *Proc. of the 14th IEEE Int. Conf. on Data Engineering (ICDE'98)*, pages 14–23, 1998.

[47] D. Florescu, A. Levy, and D. Suciu. Query containment for conjunctive queries with regular expressions. In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'98)*, pages 139–148, 1998.

[48] D. Florescu, A. Y. Levy, I. Manolescu, and D. Suciu. Query optimization in the presence of limited access patterns. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 311–322, 1999.

[49] M. Friedman, A. Levy, and T. Millstein. Navigational plans for data integration. In *Proc. of the 16th Nat. Conf. on Artificial Intelligence (AAAI'99)*, pages 67–73. AAAI Press/The MIT Press, 1999.

[50] H. Galhardas, D. Florescu, D. Shasha, and E. Simon. An extensible framework for data cleaning. Technical Report 3742, INRIA, Rocquencourt, 1999.

[51] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. D. Ullman, V. Vassalos, and J. Widom. The TSIMMIS approach to mediation: Data models and languages. *J. of Intelligent Information Systems*, 8(2):117–132, 1997.

[52] C. H. Goh, S. Bressan, S. E. Madnick, and M. D. Siegel. Context interchange: New features and formalisms for the intelligent integration of information. *ACM Trans. on Information Systems*, 17(3):270–293, 1999.

[53] G. Grahne and A. 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, 1999.

[54] G. Greco, S. Greco, and E. Zumpano. A logic programming approach to the integration, repairing and querying of inconsistent databases. In *Proc. of the 17th Int. Conf. on Logic Programming (ICLP'01)*, volume 2237 of *Lecture Notes in Artificial Intelligence*, pages 348–364. Springer, 2001.

[55] S. Gribble, A. Halevy, Z. Ives, M. Rodrig, and D. Suciu. What can databases do for peer-to-peer? In *Proc. of the Int. Workshop on the Web and Databases (WebDB'01)*, 2001.

[56] S. Grumbach, M. Rafanelli, and L. Tininini. Querying aggregate data. In *Proc. of the 18th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'99)*, pages 174–184, 1999.

[57] S. Grumbach and L. Tininini. On the content of materialized aggregate views. In *Proc. of the 19th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2000)*, pages 47–57, 2000.

[58] M. Gruninger and J. Lee. Ontology applications and design. *Communications of the ACM*, 45(2):39–41, 2002.

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

[60] A. Y. Halevy. Answering queries using views: A survey. *Very Large Database J.*, 10(4):270–294, 2001.

[61] R. Hull. Managing semantic heterogeneity in databases: A theoretical perspective. In *Proc. of the 16th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'97)*, 1997.

[62] T. Kirk, A. Y. Levy, Y. Sagiv, and D. Srivastava. The Information Manifold. In *Proceedings of the AAAI 1995 Spring Symp. on Information Gathering from Heterogeneous, Distributed Enviroments*, pages 85–91, 1995.

[63] A. C. Klug. On conjunctive queries containing inequalities. *J. of the ACM*, 35(1):146–160, 1988.

[64] D. Lembo, M. Lenzerini, and R. Rosati. Source inconsistency and incompleteness in data integration. In *Proc. of the 9th Int. Workshop on Knowledge Representation meets Databases (KRDB 2002)*, 2002.

[65] A. 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.

[66] A. Y. Levy, A. O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. In *Proc. of the 14th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'95)*, pages 95–104, 1995.

[67] A. Y. Levy, A. Rajaraman, and J. J. Ordille. Querying heterogenous information sources using source descriptions. In *Proc. of the 22nd Int. Conf. on Very Large Data Bases (VLDB'96)*, 1996.

[68] A. Y. Levy, A. Rajaraman, and J. D. Ullman. Answering queries using limited external query processors. In *Proc. of the 15th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'96)*, pages 227–237, 1996.

[69] A. Y. Levy and M.-C. Rousset. CARIN: A representation language combining Horn rules and description logics. In *Proc. of the 12th Eur. Conf. on Artificial Intelligence (ECAI'96)*, pages 323–327, 1996.

[70] A. Y. Levy and M.-C. Rousset. Combining horn rules and description logics in CARIN. *Artificial Intelligence*, 104(1–2):165–209, 1998.

[71] A. Y. Levy and D. Suciu. Deciding containment for queries with complex objects. In *Proc. of the 16th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'97)*, pages 20–31, 1997.

[72] C. Li, M. Bawa, and J. D. Ullman. Minimizing view sets without loosing query-answering power. In *Proc. of the 8th Int. Conf. on Database Theory (ICDT 2001)*, pages 99–103, 2001.

[73] C. Li and E. Chang. Query planning with limited source capabilities. In *Proc. of the 16th IEEE Int. Conf. on Data Engineering (ICDE 2000)*, pages 401–412, 2000.

[74] C. Li and E. Chang. On answering queries in the presence of limited access patterns. In *Proc. of the 8th Int. Conf. on Database Theory (ICDT 2001)*, pages 219–233, 2001.

[75] C. Li, R. Yerneni, V. Vassalos, H. Garcia-Molina, Y. Papakonstantinou, J. D. Ullman, and M. Valiveti. Capability based mediation in TSIMMIS. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 564–566, 1998.

[76] J. Lin and A. O. Mendelzon. Merging databases under constraints. *Int. J. of Cooperative Information Systems*, 7(1):55–76, 1998.

[77] J. W. Lloyd. *Foundations of Logic Programming (Second, Extended Edition)*. Springer, Berlin, Heidelberg, 1987.

[78] I. Manolescu, D. Florescu, and D. Kossmann. Answering XML queries on heterogeneous data sources. In *Proc. of the 27th Int. Conf. on Very Large Data Bases (VLDB 2001)*, pages 241–250, 2001.

[79] T. D. Millstein, A. Y. Levy, and M. Friedman. Query containment for data integration systems. In *Proc. of the 19th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2000)*, pages 67–75, 2000.

[80] T. Milo and D. Suciu. Index structures for path expressions. In *Proc. of the 7th Int. Conf. on Database Theory (ICDT'99)*, volume 1540 of *Lecture Notes in Computer Science*, pages 277–295. Springer, 1999.

[81] F. Naumann, U. Leser, and J. C. Freytag. Quality-driven integration of heterogenous information systems. In *Proc. of the 25th Int. Conf. on Very Large Data Bases (VLDB'99)*, pages 447–458, 1999.

[82] Y. Papakonstantinou and V. Vassalos. Query rewriting using semistructured views. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, 1999.

[83] E. Rahn and P. A. Bernstein. A survey of approaches to automatic schema matching. *Very Large Database J.*, 10(4):334–350, 2001.

[84] A. Rajaraman, Y. Sagiv, and J. D. Ullman. Answering queries using templates with binding patterns. In *Proc. of the 14th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'95)*, 1995.

[85] R. Reiter. On closed world data bases. In H. Gallaire and J. Minker, editors, *Logic and Databases*, pages 119–140. Plenum Publ. Co., New York, 1978.

[86] Y. Sagiv and M. Yannakakis. Equivalences among relational expressions with the union and difference operators. *J. of the ACM*, 27(4):633–655, 1980.

[87] 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.

[88] 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.

[89] J. 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, 1997.

[90] R. van der Meyden. *The Complexity of Querying Indefinite Information.* PhD thesis, Rutgers University, 1992.

[91] R. van der Meyden. Logical approaches to incomplete information. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems*, pages 307–356. Kluwer Academic Publisher, 1998.

[92] G. Zhou, R. Hull, R. King, and J.-C. Franchitti. Using object matching and materialization to integrate heterogeneous databases. In *Proc. of the 3rd Int. Conf. on Cooperative Information Systems (CoopIS'95)*, pages 4–18, 1995.

Noname manuscript No.
(will be inserted by the editor)

Answering Queries Using Views: A Survey

Alon Y. Halevy
Department of Computer Science and Engineering
University of Washington
Seattle, WA, 98195
alon@cs.washington.edu

Address(es) of author(s) should be given

Abstract The problem of answering queries using views is to find efficient methods of answering a query using a set of previously defined materialized views over the database, rather than accessing the database relations. The problem has recently received significant attention because of its relevance to a wide variety of data management problems. In query optimization, finding a rewriting of a query using a set of materialized views can yield a more efficient query execution plan. To support the separation of the logical and physical views of data, a storage schema can be described using views over the logical schema. As a result, finding a query execution plan that accesses the storage amounts to solving the problem of answering queries using views. Finally, the problem arises in data integration systems, where data sources can be described as precomputed views over a mediated schema. This article surveys the state of the art on the problem of answering queries using views, and synthesizes the disparate works into a coherent framework. We describe the different applications of the problem, the algorithms proposed to solve it and the relevant theoretical results.

1 Introduction

The problem of answering queries using views (a.k.a. rewriting queries using views) has recently received significant attention because of its relevance to a wide variety of data management problems: query optimization, maintenance of physical data independence, data integration and data warehouse design. Informally speaking, the problem is the following. Suppose we are given a query Q over a database schema, and a set of view definitions $V_1, \ldots, V_n$ over the same schema. Is it possible to answer the query Q using *only* the answers to the views $V_1, \ldots, V_n$? Alternatively, what is the maximal set of tuples in the answer of Q that we can obtain from the views? If we can access both the views and the database relations, what is the cheapest query execution plan for answering Q?

The first class of applications in which we encounter the problem of answering queries using views is query optimization and database design. In the context of query optimization, computing a query using previously materialized views can speed up query processing because part of the computation necessary for the query may have already been done while computing the views. Such savings are especially significant in decision support applications when the views and queries contain grouping and aggregation. Furthermore, in some cases, certain indices can be modeled as precomputed views (e.g., join indices [Val87]),[1] and deciding which indices to use requires a solution to the query rewriting problem. In the context of database design, view definitions provide a mechanism for supporting the independence of the *physical* view of the data and its *logical* view. This independence enables us to modify the storage schema of the data (i.e., the physical view) without changing its logical schema, and to model more complex types of indices. Hence, several

[1] Strictly speaking, to model join indices we need to extend the logical model to refer to row IDs.

authors describe the storage schema as a set of views over the logical schema [YL87,TSI96,Flo96]. Given these descriptions of the storage, the problem of computing a query execution plan (which, of course, must access the physical storage) involves figuring out how to use the views to answer the query.

A second class of applications in which our problem arises is data integration. Data integration systems provide a uniform query interface to a multitude of autonomous data sources, which may reside within an enterprise or on the World-Wide Web. Data integration systems free the user from having to locate sources relevant to a query, interact with each one in isolation, and manually combine data from multiple sources. Users of data integration systems do not pose queries in terms of the schemas in which the data is stored, but rather in terms of a *mediated schema*. The mediated schema is a set of relations that is designed for a specific data integration application, and contains the salient aspects of the domain under consideration. The tuples of the mediated schema relations are not actually stored in the data integration system. Instead, the system includes a set of *source descriptions* that provide semantic mappings between the relations in the source schemas and the relations in the mediated schema.

The data integration systems described in [LRO96b,DG97b,KW96,LKG99] follow an approach in which the contents of the sources are described as views over the mediated schema. As a result, the problem of reformulating a user query, posed over the mediated schema, into a query that refers directly to the source schemas becomes the problem of answering queries using views. In a sense, the data integration context can be viewed as an extreme case of the need to maintain physical data independence, where the logical and physical layout of the data sources has been defined in advance. The solutions to the problem of answering queries using views differ in this context because the number of views (i.e., sources) tends to be much larger, and the sources need not contain the *complete* extensions of the views.

In the area of data warehouse design we need to choose a set of views (and indexes on the views) to materialize in the warehouse [HRU96,TS97,YKL97,GHRU97,ACN00,CG00]. Similarly, in web-site design, the performance of a web site can be significantly improved by choosing a set of views to materialize [FLSY99]. In both of these problems, the first step in determining the utility of a choice of views is to ensure that the views are sufficient for answering the queries we expect to receive over the data warehouse or the web site. This problem, again, translates into the view rewriting problem.

Finally, answering queries using views plays a key role in developing methods for semantic data caching in client-server systems [DFJ$^+$96,KB96,CR94,ACPS96]. In these works, the data cached at the client is modeled semantically as a set of queries, rather than at the physical level as a set of data pages or tuples. Hence, deciding which data needs to be shipped from the server in order to answer a given query requires an analysis of which parts of the query can be answered by the cached views.

The many applications of the problem of answering queries using views has spurred a flurry of research, ranging from theoretical foundations to algorithm design and implementation in several commercial systems. This article surveys the current state of the art in this area, and classifies the works into a coherent framework based on a set of dimensions along which the treatments of the problem differ.

The treatments of the problem differ mainly depending on whether they are concerned with query optimization and database design or with data integration. In the case of query optimization and database design, the focus has been on producing a query execution plan that involves the views, and hence the effort has been on extending query optimizers to accommodate the presence of views. In this context, it is necessary that rewriting of the query using the views be an *equivalent* rewriting in order for the query execution plan to be correct. It is important to note that some of the views included in the query plan may not contribute to the logical correctness of the plan, but only to reducing the plan's cost.

In the data integration context, the focus has been on translating queries formulated in terms of a mediated schema into queries formulated in terms of data sources. Hence, the output of the algorithm is a query expression, rather than a query execution plan. Because the data sources may not entirely cover the domain, we sometimes need to settle for a *contained* query rewriting, rather than an equivalent one. A contained query rewriting provides a subset of the answer to the query, but perhaps not the entire answer. In addition, the works on data integration distinguished between the case in which the individual views are complete (i.e., contain all the tuples in their definition) and the case where they may be incomplete (as is common when modeling autonomous data sources). Furthermore, the works on data integration distinguished the translation problem from the more general problem of finding all the answers to a query given the data in the sources, and showed that the two problems differ in interesting ways.

The survey is organized as follows. Section 2 presents in more detail the applications motivating the study of the problem and the dimensions along which we can study the problem. Section 3 defines the problem formally. As a basis for the discussion of the different algorithms, Section 4 provides an intuitive explanation of the conditions under which a view can be used to answer a query. Section 5 describes how materialized views have been incorporated into query optimization. Section 6 describes algorithms for answering queries using views that were developed in the context of data integration. Section 7 surveys some theoretical issues concerning the problem of answering queries using views, and Section 8 discusses several extensions to the algorithms in Sections 5 and 6 to accommodate queries over object-oriented databases and queries with access-pattern limitations. Finally, Section 9 concludes, and outlines some of the open problems in this area.

We note that this survey is not concerned with the closely related problems of incremental maintenance of materialized views, which is surveyed in [GM99b], selection of which views to maintain in a data warehouse [HRU96, TS97, GHRU97, Gup97b, YKL97, GM99c, CG00, CHS01] or automated selection of indexes [CN98b, CN98a].

2 Motivation and Illustrative Examples

Before beginning the detailed technical discussion, we motivate the problem of answering queries using views through some of its applications. In particular, this section serves to illustrate the wide and seemingly disparate range of applications of the problem. We end the section by classifying the different works on the topic into a taxonomy.

We use the following familiar university schema in our examples throughout the paper. We assume that professors and students and departments are uniquely identified by their names, and courses are uniquely identified by their numbers. The Registered relation describes the students' registration in classes, while the Major relation describes in which department a particular student is majoring (we assume for simplicity that every department has a single major program).

Prof(name, area)
Course(c-number, title)
Teaches(prof, c-number, quarter, evaluation)
Registered(student, c-number, quarter)
Major(student, dept)
WorksIn(prof, dept)
Advises(prof, student).

2.1 Query Optimization

The first and most obvious motivation for considering the problem of answering queries using views is for query optimization. If part of the computation needed to answer a query has already been performed in computing a materialized view, then we can use the view to speed up the computation of the query.

Consider the following query, asking for students and course titles for students who registered in Ph.D-level classes taught by professors in the Database area (in our example university graduate level classes have numbers of 400 and above, and Ph.D-level courses numbers of 500 and above):

```
select   Registered.student, Course.title
from     Teaches, Prof, Registered, Course
where    Prof.name=Teaches.prof and Teaches.c-number=Registered.c-number and
         Teaches.quarter=Registered.quarter and Registered.c-number=Course.c-number and
         Course.c-number ≥ 500 and Prof.area="DB".
```

Suppose we have the following materialized view, containing the registration records of graduate level courses and above.

```
create view Graduate as
select    Registered.student, Course.title, Course.c-number, Registered.quarter
from      Registered, Course
where     Registered.c-number=Course.c-number and Course.c-number ≥ 400.
```

The view Graduate can be used in the computation of the above query as follows:

```
select    Graduate.student, Graduate.title
from      Teaches, Prof, Graduate
where     Prof.name=Teaches.prof and
          Teaches.c-number=Graduate.c-number and Teaches.quarter=Graduate.quarter and
          Graduate.c-number ≥ 500 and Prof.area="DB".
```

The resulting evaluation will be cheaper because the view Graduate has already performed the join between Registered and Course, and has already pruned the non-graduate courses (the courses that actually account for most of the activity going on in a typical university). It is important to note that the view Graduate is useful for answering the query even though it does not *syntactically* match any of the subparts of the query.

Even if a view has already computed part of the query, it is not necessarily the case that using the view will lead to a more efficient evaluation plan, especially considering the indexes available on the database relations and on the views. For example, suppose the relations Course and Registered have indexes on the c-number attribute. In this case, if the view Graduate does not have any indexes, then evaluating the query directly from the database relations may be cheaper. Hence, the challenge is not only to detect when a view is logically usable for answering a query, but also to make a judicious cost-based decision on when to use the available views.

2.2 Maintaining Physical Data Independence

Several works on answering queries using views were inspired by the goal of maintaining physical data independence in relational and object-oriented databases [YL87,TSI96,Flo96]. One of the principles underlying modern database systems is the separation between the logical view of the data (e.g., as tables with their named attributes) and the physical view of the data (i.e., how it is laid out on disk). With the exception of horizontal or vertical partitioning of relations into multiple files, relational database systems are still largely based on a 1-1 correspondence between relations in the schema and files in which they are stored. In object-oriented systems, maintaining the separation is necessary because the logical schema contains significant redundancy, and does not correspond to a good physical layout. Maintaining physical data independence becomes more crucial in applications where the logical model is introduced as an intermediate level after the physical representation has already been determined. This is common in applications of semi-structured data [Bun97,Abi97,FLM98], storage of XML data in relational databases [FK99,SGT$^+$99, DFS99,TIHW01], and in data integration. In fact, the STORED System [DFS99] stores XML documents in a relational database, and uses views to describe the mapping from XML into relations in the database. In some sense, data integration, discussed in the next section, is an extreme case where there is a separation between the logical view of the data and its physical view.

To maintain physical data independence, several authors proposed to use views as a mechanism for describing the storage of the data. In particular, [TSI96] described the storage of the data using GMAPs *(generalized multi-level access paths)*, expressed over the conceptual model of the database.

To illustrate, consider the entity-relationship model of a slightly extended university domain shown in Figure 1. Figure 2 shows GMAPs expressing the different storage structures for this data.

A GMAP describes the physical organization and indexes of the storage structure. The first clause of the GMAP (the as clause) describes the actual data structure used to store a set of tuples (e.g., a B$^+$-tree, hash index, etc.) The remaining clauses describe the content of the structure, much like a view definition. The given and select clauses describe the available attributes, where the given clause describes the attributes on which the structure is indexed. The definition of the view, given in the where clause uses infix notation over the conceptual model.

In our example, the GMAP G1 stores a set of pairs containing students and the departments in which they major, and these pairs are indexed by a B$^+$-tree on attribute Student.name. The GMAP G2 stores an

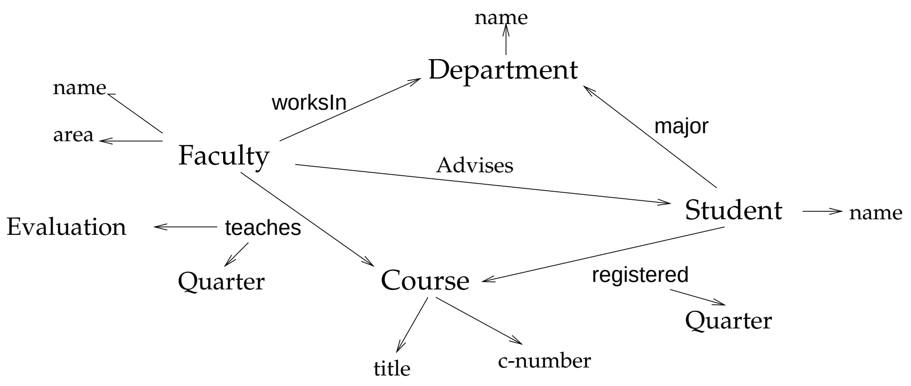

Fig. 1 An Entity/Relationship diagram for the university domain. Note that quarter is an attribute of the relationships registered and teaches.

```
def_gmap G1 as b+-tree by                def_gmap G2 as b+-tree by
    given Student.name                       given Student.name
    select Department                        select Course.c-number
    where Student major Department.          where Student registered Course.

def_gmap G3 as b+-tree by
    given Course.c-number
    select Department
    where Student registered Course and Student major Department.
```

Fig. 2 GMAPs for the university domain.

index from the names of students to the numbers of the courses in which they are registered. The GMAP G3 stores an index from course numbers to departments whose majors are enrolled in the course. As shown in [TSI96], using GMAPs it is possible to express a large family of data structures, including secondary indexes on relations, nested indexes, collection based indexes and structures implementing field replication.

Given that the data is stored in the structures described by the GMAPs, the question that arises is how to use these structures to answer queries. Since the logical content of the GMAPs are described by views, answering a query amounts to finding a way of rewriting the query using these views. If there are multiple ways of answering the query using the views, we would like to find the cheapest one. Note that in contrast to the query optimization context, we *must* use the views to answer a given query, because all the data is stored in the GMAPs,

Consider the following query in our domain, which asks for names of students registered for Ph.D-level courses and the departments in which these students are majoring.

```
select   Student.name, Department
where    Student registered Course and Student major Department and Course.c-number≥500.
```

The query can be answered in two ways. First, since Student.name uniquely identifies a student, we can take the join of G1 and G2, and then apply a selection Course.c-number≥500, and a projection on Student.name and Department. A second solution would be to join G3 with G2 and select Course.c-number≥500. In fact, this solution may even be more efficient because G3 has an index on the course number and therefore the intermediate joins may be much smaller.

2.3 Data Integration

Much of the recent work on answering queries using views has been spurred because of its applicability to data integration systems. A data integration system (a.k.a. a mediator system [Wie92]) provides a *uniform* query interface to a multitude of autonomous heterogeneous data sources. Prime examples of data integration applications include enterprise integration, querying multiple sources on the World-Wide Web, and integration of data from distributed scientific experiments. The sources in such an application may be traditional databases, legacy systems, or even structured files. The goal of a data integration system is to free the user from having to find the data sources relevant to a query, interact with each source in isolation, and manually combine data from the different sources.

To provide a uniform interface, a data integration system exposes to the user a *mediated schema.* A mediated schema is a set of *virtual* relations, in the sense that they are not actually stored anywhere. The mediated schema is designed manually for a particular data integration application. To be able to answer queries, the system must also contain a set of *source descriptions.* A description of a data source specifies the contents of the source, the attributes that can be found in the source, and the constraints on the contents of the source.

One of the approaches for specifying source descriptions, which has been adopted in several systems ([LRO96b,KW96,FW97,DG97b,LKG99]), is to describe the contents of a data source as a *view* over the mediated schema. This approach facilitates the addition of new data sources and the specification of constraints on contents of sources (see [Ull97,FLM98,Lev00] for a comparison of different approaches for specifying source descriptions).

In order to answer a query, a data integration system needs to translate a query formulated on the mediated schema into one that refers directly to the schemas in the data sources. Since the contents of the data sources are described as views, the translation problem amounts to finding a way to answer a query using a set of views.

We illustrate the problem with the following example, where the mediated schema exposed to the user is our university schema, except that the relations Teaches and Course have an additional attribute identifying the university at which a course is being taught:

```
Teaches(prof, c-number, quarter, evaluation, univ)
Course(c-number, title, univ)
```

Suppose we have the following two data sources. The first source provides a listing of all the courses titled "Database Systems" taught anywhere and their instructors. This source can be described by the following view definition:

```
create view DB-courses as
select    Course.title, Teaches.prof, Course.c-number, Course.univ
from      Teaches, Course
where     Teaches.c-number=Course.c-number and Teaches.univ=Course.univ and
          Course.title="Database Systems".
```

The second source lists Ph.D level courses being taught at the University of Washington (UW), and is described by the following view definition:

```
create view UW-phd-courses as
select    Course.title, Teaches.prof, Course.c-number, Course.univ
from      Teaches, Course
where     Teaches.c-number=Course.c-number and
          Course.univ="UW" and Teaches.univ="UW" and Course.c-number≥500.
```

If we were to ask the data integration system who teaches courses titled "Database Systems" at UW, it would be able to answer the query by applying a selection on the source DB-courses:

```
select    prof
from      DB-courses
where     univ="UW".
```

On the other hand, suppose we ask for all the graduate-level courses (not just in databases) being offered at UW. Given that only these two sources are available, the data integration system cannot find *all* tuples in the answer to the query. Instead, the system can attempt to find the maximal set of tuples in the answer that are available from the sources. In particular, the system can obtain graduate *database* courses at UW from the DB-courses source, and the Ph.D level courses at UW from the UW-Phd-courses source. Hence, the following query provides the maximal set of answers that can be obtained from the two sources:

```
select    title, c-number
from      DB-courses
where     univ="UW" and c-number≥400
       UNION
select    title, c-number
from      UW-phd-courses.
```

Note that courses that are not Ph.D-level courses or database courses will not be returned as answers. Whereas in the contexts of query optimization and maintaining physical data independence the focus is on finding a query expression that is *equivalent* to the original query, here we attempt to find a query expression that provides the *maximal answers* from the views. We formalize both of these notions in Section 3.

Other applications: Before proceeding, we also note that the problem of answering queries using views arises in the design of data warehouses (e.g., [HRU96,TS97,GHRU97,YKL97]) and in semantic data caching. In data warehouse design, when we choose a set of views to materialize in a data warehouse, we need to check that we will be able to answer all the required queries over the warehouse using only these views. In the context of semantic data caching (e.g., [DFJ$^+$96,KB96,CR94,ACPS96]) we need to check whether the cached results of a previously computed query can be used for a new query, or whether the client needs to request additional data from the server. In [FLSY99,YFIV00] it is shown that precomputing views can significantly speed up the response time from web sites, which again raises the question of view selection.

2.4 A taxonomy of the field

As illustrated by the examples, there are several dimensions along which we can classify the treatments of the problem of answering queries using views. In this section we describe a taxonomy for classifying the different works on this problem, and highlight the main differences between the problem treatments. Figure 3 shows the taxonomy and some of the representative works belonging to each of its classes.

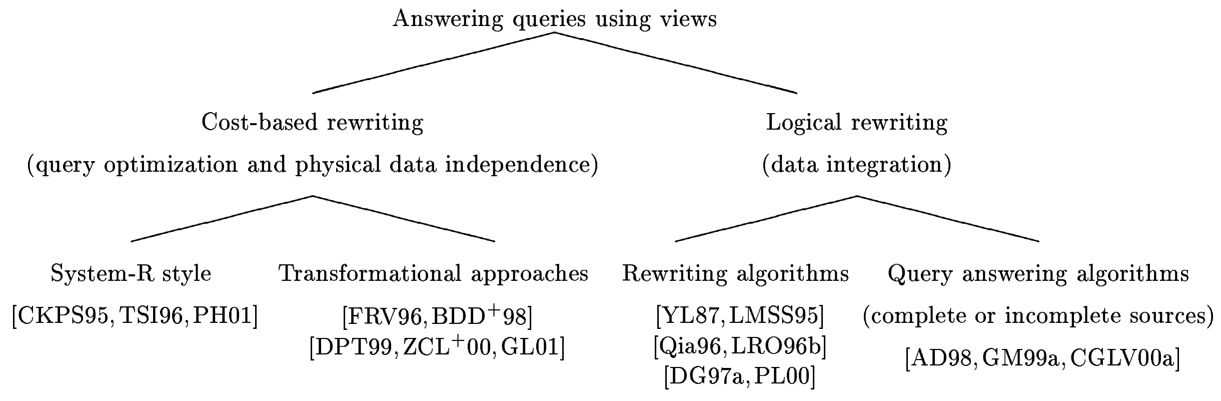

Fig. 3 A taxonomy of work on answering queries using views. The main distinction is between works on query optimization and maintenance of physical data independence and works considering logical rewritings, mostly in the context of data integration. The works on query optimization have considered both System-R style algorithms and transformation-based algorithms. The works on data integration considered algorithms that scale to a large number of views, and the question of finding all the answers to the query, given the view extensions.

The most significant distinction between the different works is whether their goal is data integration or whether it is query optimization and maintenance of physical data independence. The key difference between these two classes of works is the output of the algorithm for answering queries using views. In the former case, given a query Q, and a set of views $\mathcal{V}$, the goal of the algorithm is to produce an expression Q' that references the views and is either equivalent to or contained in Q. In the latter case, the algorithm must go further and produce a (hopefully optimal) query execution plan for answering Q using the views (and possibly the database relations). Here the rewriting must be an equivalent to Q in order to ensure the correctness of the plan.

The similarity between these two bodies of work is that they are concerned with the core issue of whether a rewriting of a query is equivalent to or contained in the query. However, while logical correctness suffices for the data integration context, it does not in the query optimization context where we also need to find the *cheapest* plan using the views. The complication arises because the optimization algorithms need to consider views that do not contribute to the *logical* correctness of the rewriting, but do reduce the cost of the resulting plan. Hence, while the reasoning underlying the algorithms in the data integration context is mostly logical, in the query optimization case it is both logical and cost-based. On the other hand, an aspect stressed in the data integration context is the importance of dealing with a large number of views, which correspond to data sources. In the context of query optimization it is generally assumed (not always!) that the number of views is roughly comparable to the size of the schema.

Extension	Relevant works
Grouping and aggregation	[GHQ95, SDJL96, CNS99, GRT99, ZCL$^+$00, GT00] (Section 5.3)
Bag semantics	[CKPS95, ZCL$^+$00] (Section 5.3)
OQL	[FRV96, DPT99] (Section 8.1)
Multi-block queries	[ZCL$^+$00] (Section 5.2)
Integrity constraints	[DL97, Gry98, ZCL$^+$00, DPT99] (Section 7.2)
Access-pattern limitations	[RSU95, KW96, DL97] (Section 8.2)
Unions in the views	[AGK99, Dus98] (Section 8.3)
Queries over semi-structured data	[CGLV99, PV99] (Section 8.3)
Hierarchies in Description Logics	[BLR97, CGL99] (Section 8.3)
Languages for querying schema	[Mil98] (Section 8.3)

Table 1 Extensions to query and view languages

The works on query optimization can be classified into System-R style optimizers and transformational optimizers. The initial works incorporated views into System-R style join enumeration, while later works that attempt to deal with a more extended subset of SQL realized that the power of rewriting rules is required in order to incorporate views.

The main line of work on data integration attempted to develop algorithms for answering queries using views that scale up to a large number of views[2]. A second line of work started considering different properties of the data sources. For example, it was shown that if data sources are assumed to be complete (i.e., they include all the tuples that satisfy their definition), then the problem of answering queries using views becomes computationally harder. Intuitively, the reason for the added complexity is that when sources are complete, we can also infer negative information as a result of a query to the source. This led to asking the following more basic question: given a query Q, a set of views $\mathcal{V}$ *and* their extensions, what is the complexity of finding the maximal set of tuples in the answer to Q from $\mathcal{V}$.[3] This work established an interesting connection between the problem of answering queries using views and query answering in conditional tables [IL84]. In these works, a major factor affecting the complexity of the problem is whether the view extensions are assumed to be complete or not (when they are complete, the complexity is higher). Note that in the context of query optimization, the views are always assumed to be complete.

A separate dimension for classifying the different works is the specific language used for expressing views and queries. Much of the early work on the problem focused on select-project-join queries, but, as shown in Table 1, many extensions have been considered as well. The works on query optimization have considered

[2] Strictly speaking, the motivation for the work of [YL87] was the maintenance of physical data independence, but their algorithm has more similarities with the data integration algorithms.

[3] Some authors refer to the distinction between the two problems as the *rewriting* problem versus the *query answering* problem.

extensions of interest to SQL engines, such as grouping and aggregation and the presence of certain integrity constraints on the database relations. For obvious reasons, these works have also considered the implications of bag semantics on the rewriting problem. The data integration works have considered extensions such as access-pattern limitation to the views, recursive queries, path expressions in the queries, and integrity constraints expressed in description logics.

3 Problem Definition

In this section we define the basic terminology used throughout this paper. We define the concepts of query containment and query equivalence that provide a semantic basis for comparing between queries and their rewritings, and then define the problem of answering queries using views. Finally, we define the problem of extracting *all* the answers to a query from a set of views (referred to as the set of certain answers).

The bulk of our discussion will focus on the class of select-project-join queries on relational databases. A view is a named query. It is said to be materialized if its results are stored in the database. A database instance is an assignment of an extension (i.e., a set of tuples) to each of the relations in the database.

We assume the reader is familiar with the basic elements of SQL. We will distinguish between queries that involve arithmetic comparison predicates (e.g., $\leq, <, \neq$) and those that do not. Our discussion of answering queries using views in the context of data integration systems will require considering recursive datalog queries. We recall the basic concepts of datalog in Section 6.

In our discussion, we denote the result of computing the query Q over the database D by $Q(D)$. We often refer to queries that reference named views (e.g., in query rewritings). In that case, $Q(D)$ refers to the result of computing Q after the views have been computed from D.

3.1 Containment and Equivalence

The notions of query containment and query equivalence enable comparison between different reformulations of queries. They will be used when we test the correctness of a rewriting of a query in terms of a set of views. In the definitions below we assume the answers to queries are sets of tuples. The definitions can be extended in a straightforward fashion to bag semantics. In the context of our discussion it is important to note that the definitions below also apply to queries that may reference named views.

Definition 1 *Query containment and equivalence: A query Q_1 is said to be contained in a query Q_2, denoted by $Q_1 \sqsubseteq Q_2$, if for all database instances D, the set of tuples computed for Q_1 is a subset of those computed for Q_2, i.e., $Q_1(D) \subseteq Q_2(D)$. The two queries are said to be equivalent if $Q_1 \sqsubseteq Q_2$ and $Q_2 \sqsubseteq Q_1$.*

The problems of query containment and equivalence have been studied extensively in the literature and should be a topic of a specialized survey. Some of the cases which are most relevant to our discussion include: containment of select-project-join queries and unions thereof [CM77,SY81], queries with arithmetic comparison predicates [Klu88,LS93,ZO93,KMT98], recursive queries [Shm93,Sag88,LS93,CV92,CV94], and queries with bag semantics [CV93]

3.2 Rewriting of a Query Using Views

Given a query Q and a set of view definitions $V_1, \ldots, V_m$, a rewriting of the query using the views is a query expression Q' that refers *only* to the views $V_1, \ldots, V_m$.[4] In SQL, a query refers only to the views if all the relations mentioned in the from clauses are views. In practice, we may also be interested in rewritings that can also refer to the database relations. Conceptually, rewritings that refer to the database relations do not introduce new difficulties, because we can always simulate the previous case by inventing views that mirror precisely the database tables.

As we saw in Section 2, we need to distinguish between two types of query rewritings: *equivalent rewritings* and *maximally-contained rewritings*. For query optimization and maintaining physical data independence we consider equivalent rewritings.

[4] Note that rewritings that refer only to the views were called *complete rewritings* in [LMSS95].

Definition 2 *Equivalent rewritings: Let Q be a query and $\mathcal{V} = \{V_1, \ldots, V_m\}$ be a set of view definitions. The query Q' is an equivalent rewriting of Q using $\mathcal{V}$ if:*
- *Q' refers only to the views in $\mathcal{V}$, and*
- *Q' is equivalent to Q.*

In the context of data integration, we often need to consider maximally-contained rewritings. Unlike the case of equivalent rewritings, the maximally-contained rewriting may differ depending on the query language we consider for the rewriting. Hence, the following definition depends on a particular query language:

Definition 3 *Maximally-contained rewritings: Let Q be a query, $\mathcal{V} = \{V_1, \ldots, V_m\}$ be a set of view definitions, and $\mathcal{L}$ be a query language. The query Q' is a maximally-contained rewriting of Q using $\mathcal{V}$ w.r.t. $\mathcal{L}$ if:*
- *Q' is a query in $\mathcal{L}$ that refers only to the views in $\mathcal{V}$,*
- *Q' is contained in Q, and*
- *there is no rewriting $Q_1 \in \mathcal{L}$, such that $Q' \sqsubseteq Q_1 \sqsubseteq Q$ and Q_1 is not equivalent to Q'.*

When a rewriting Q' is contained in Q but is not a maximally-contained rewriting we refer to it as a contained rewriting. Note that the above definitions are independent of the particular query language we consider. Furthermore, we note that algorithms for query containment and equivalence provide methods for *testing* whether a candidate rewriting of a query is an equivalent or contained rewriting. However, by themselves, these algorithms do not provide a solution to the problem of answering queries using views.

A more fundamental question we can consider is how to find *all* the possible answers to the query, given a set of view definitions and their extensions. Finding a rewriting of the query using the views and then evaluating the rewriting over the views is clearly one candidate algorithm. If the rewriting is equivalent to the query, then we are guaranteed to find all the possible answers. However, as we see in Section 7, a maximally-contained rewriting of a query using a set of views does not always provide all the possible answers that can be obtained from the views. Intuitively, the reason for this is that a rewriting is maximally-contained only w.r.t. a specific query language, and hence there may sometimes be a query in a more expressive language that may provide more answers.

The problem of finding all the answers to a query given a set of views is formalized below by the notion of *certain answers*, originally introduced in [AD98]. In the definition, we distinguish the case in which the view extensions are assumed to be complete (closed-world assumption) from the case in which the views may be partial (open-world).

Definition 4 *Certain answers: Let Q be a query and $\mathcal{V} = \{V_1, \ldots, V_m\}$ be a set of view definitions over the database schema $R_1, \ldots, R_n$. Let the sets of tuples $v_1, \ldots, v_m$ be extensions of the views $V_1, \ldots, V_m$, respectively.*

The tuple a is a certain answer *to the query Q under the closed-world assumption given $v_1, \ldots, v_m$ if $a \in Q(D)$ for all database instances D such that $V_i(D) = v_i$ for every i, $1 \leq i \leq m$.*

The tuple a is a certain answer *to the query Q under the open-world assumption given $v_1, \ldots, v_m$ if $a \in Q(D)$ for all database instances D such that $V_i(D) \supseteq v_i$ for every i, $1 \leq i \leq m$.*

The intuition behind the definition of certain answers is the following. The extensions of a set of views do not define a unique database instance. Hence, given the extensions of the views we have only partial information about the real state of the database. A tuple is a certain answer of the query Q if it is an answer for *any* of the possible database instances that are consistent with the given extensions of the views. Section 7.3 considers the complexity of finding certain answers.

Example 1 As a very simple example, consider a database schema $R(A, B)$ that includes a single relation with two attributes. Suppose the view V_1 is defined to be the projection of R on A, while V_2 is defined to be the projection of R on B, and suppose that our query Q is to retrieve all of the relation R.

Suppose we are given that the extension of V_1 includes the single tuple (c_1), and that the extension of V_2 includes the single tuple (c_2),

Under the closed-world assumption, we can infer that the tuple (c_1, c_2) *must* be in the relation R, and hence it is a certain answer to Q. However, under the open-world assumption, since V_1 and V_2 are not necessarily complete, the tuple (c_1, c_2) need not be in R. For example, R may contain the tuples (c_1, d) and (e, c_2) for some constants d and e. Hence, (c_1, c_2) is not a certain answer to Q. $\Box$

4 When is a View Usable for a Query

The common theme across all of the works on answering queries using views is that they all have to deal with the fundamental question of when a view is usable to answer a query. Hence, before describing the actual algorithms for answering queries using views it is instructive to examine a few examples and gain an intuition for the conditions under which a view is usable for answering a query, and in what ways a view may be useful. In this section we consider select-project-join queries under set semantics. Note that in some cases a view may be usable in maximally-contained rewritings but not in equivalent rewritings.

Informally, a view can be useful for a query if the set of relations it mentions overlaps with that of the query, and it selects some of the attributes selected by the query. Moreover, if the query applies predicates to attributes that it has in common with the view, then the view must apply either equivalent or logically weaker predicates in order to be part of an equivalent rewriting. If the view applies a logically stronger predicate, it may be part of a contained rewriting.

Consider the following query, asking for the triplets of professors, students, and teaching quarters, where the student is advised by the professor, and has taken a class taught by the professor during the winter of 1998 or later.

```
select  Advises.prof, Advises.student, Registered.quarter
from    Registered, Teaches, Advises
where   Registered.c-number=Teaches.c-number and Registered.quarter=Teaches.quarter and
        Advises.prof=Teaches.prof and Advises.student=Registered.student and
        Registered.quarter ≥ "winter98".
```

The following view V_1 is usable because it applies the same join conditions to the relations Registered and Teaches. Hence, we can use V_1 to answer the query by joining it with the relation Advises. Furthermore, V_1 selects the attributes Registered.student, Registered.quarter and Teaches.prof that are needed for the join with the relation Advises and for the select clause of the query. Finally, V_1 applies a predicate Registered.quarter > "winter97" which is weaker than the predicate Registered.quarter ≥ "winter98" in the query. However, since V_1 selects the attribute Registered.quarter, the stronger predicate can be applied as part of the rewriting.

```
create view V1 as
select  Registered.student, Teaches.prof, Registered.quarter
from    Registered, Teaches
where   Registered.c-number=Teaches.c-number and Registered.quarter=Teaches.quarter and
        Registered.quarter > "winter97".
```

The views shown in Figure 4 illustrate how minor modifications to V_1 change their usability in answering the query. The view V_2 is similar to V_1, except that it does not select the attribute Teaches.prof, which is needed for the join with the relation Advises and in the select clause of the query. Hence, to use V_2 in the rewriting, we would need to join V_2 with the Teaches relation again (in addition to a join with Advises). Still, if the join of the relations Registered and Teaches is very selective, then employing V_2 may actually result in a more efficient query execution plan.

The view V_3 does not apply the necessary equi-join predicate between Registered.quarter and Teaches.quarter. Since the attributes Teaches.quarter and Registered.quarter are not selected by V_3, the join predicate cannot be applied in the rewriting, and therefore there is little to gain by using V_3. The view V_4 considers only the professors who have at least one area of research. Hence, the view applies an additional condition that does not exist in the query, and cannot be used in an equivalent rewriting unless we allow union and negation in the rewriting language. However, if we have an integrity constraint stating that every professor has at least one area of research, then an optimizer should be able to realize that V_4 is usable. Finally, view V_5 applies a stronger predicate than in the query (Registered.quarter > "winter99"), and is therefore usable for a contained rewriting, but not for an equivalent rewriting of the query.

To summarize, the following conditions need to hold in order for a select-project-join view V to be usable in an equivalent rewriting of a query Q. The intuitive conditions below can be made formal in the context of a specific query language and/or available integrity constraints (see e.g., [YL87,LMSS95]).

```
create view V2 as
select  Registered.student, Registered.quarter
from    Registered, Teaches
where   Registered.c-number=Teaches.c-number
and     Registered.quarter=Teaches.quarter
and     Registered.quarter ≥ "winter98".
```

```
create view V3 as
select  Registered.student, Teaches.prof, Registered.quarter
from    Registered, Teaches
where   Registered.c-number=Teaches.c-number
and     Registered.quarter ≥ "winter98".
```

```
create view V4 as
select  Registered.student, Registered.quarter,
        Teaches.prof
from    Registered, Teaches, Advises, Area
where   Registered.c-number=Teaches.c-number
and     Registered.quarter=Teaches.quarter
and     Teaches.prof=Advises.prof
and     Teaches.prof=Area.name
and     Registered.quarter ≥ "winter98"
```

```
create view V5 as
select  Registered.student, Teaches.prof, Registered.quarter

from    Registered, Teaches
where   Registered.c-number=Teaches.c-number
and     Registered.quarter=Teaches.quarter
and     Registered.quarter > "winter99".
```

Fig. 4 Examples of unusable views.

1. There must be a mapping ψ from the occurrences of tables mentioned in the from clause of V to those mentioned in the from clause of Q, mapping every table name to itself. In the case of bag semantics, ψ must be a 1-1 mapping, whereas for set semantics, ψ can be a many-to-1 mapping.
2. V must either apply the join and selection predicates in Q on the attributes of the tables in the domain of ψ, or must apply to them a logically weaker selection, and select the attributes on which predicates need to still be applied.
3. V must not project out any attributes of the tables in the domain of ψ that are needed in the selection of Q, unless these attributes can be recovered from another view (or from the original table if it's available).

Finally, we note that the introduction of bag semantics introduces additional subtleties. In particular, we must ensure that the multiplicity of answers required in the query are not lost in the views (e.g., by the use of distinct), and are not increased (e.g., by the introduction of additional joins).

5 Incorporating Materialized Views into Query Optimization

This section describes the different approaches to incorporating materialized views into query optimization. The focus of these algorithms is to judiciously decide when to use views to answer a query. The output of the algorithm is an execution plan for the query. The approaches differ depending on which phase of query optimization was modified to consider materialized views. Section 5.1 describes algorithms based on System R-style optimization, where materialized views are considered during the join enumeration phase [CKPS95, TSI96]. Section 5.2 describes works based on transformational optimizers [ZCL+00,DPT99,PDST00,GL01]. There, the key idea is that replacing a query subexpression by a view is yet another transformation employed by the optimizer. Section 5.3 discusses some of the issues that arise when rewriting algorithms are extended to consider grouping and aggregation. These extensions are key to incorporating materialized views into decision support applications.

5.1 System-R style optimization

In this section we consider select-project-join queries and discuss the changes that need to be made to a join enumeration algorithm to incorporate materialized views. To illustrate the changes to a System R-style optimizer we first briefly recall the principles underlying System-R optimization [SAC+79]. System-R takes a bottom-up approach to building query execution plans. In the first phase, it constructs plans of size 1, i.e., chooses the best access paths to every table mentioned in the query. In phase n, the algorithm considers plans of size n, by combining pairs of plans obtained in the previous phases (Note that if the algorithm is considering only left-deep plans, it will try to combine plans of size $n-1$ with plans of size 1. Otherwise, it will consider combining plans of size k with plans of size $n-k$.) The algorithm terminates after constructing plans that cover all the relations in the query.

Intuitively, the efficiency of System-R stems from the fact that it partitions query execution plans into *equivalence classes*, and only considers a single execution plan for every equivalence class. Two plans are in the same equivalence class if they (1) cover the same set of relations in the query (and therefore are also of the same size), and (2) produce the answers in the same interesting order. In the process of building plans, two plans are combined only if they cover disjoint subsets of the relations mentioned in the query.

In our context, the query optimizer builds query execution plans by accessing a set of views, rather than a set of database relations. Hence, in addition to the meta-data that the query optimizer has about the materialized views (e.g., statistics, indexes) the optimizer is also given as input the query expressions defining the views. Recall that a database relation can always be modeled as a view as well.

We illustrate the changes to the join enumeration algorithm with an example that includes the following views:

```
create view V1 as
select    student, dept
from      Major.

create view V2 as
select    Registered.student, Registered.c-number
from      Registered, Course
where     Registered.c-number=Course.c-number
          and Course.title LIKE '%theory%'.

create view V3 as
select    Major.dept, Registered.c-number
from      Registered, Major
where     Registered.student=Major.student and Registered.c-number≥500.
```

Suppose the query below asks for all of the students attending Ph.D level classes with 'theory' in their title, and the departments in which the students are majoring.

```
select    Registered.student, Major.dept
from      Registered, Major, Course
where     Registered.student=Major.student and Registered.c-number=Course.c-number and
          Course.c-number≥500 and Course.title LIKE '%theory%'.
```

We now describe the additional issues that the optimizer needs to consider in the presence of materialized views. Figure 5 shows a side-by-side comparison of the steps of a traditional optimizer vs. one that exploits materialized views. The algorithm described below is a slight modification of the GMAP algorithm [TSI96]. The algorithm described in [CKPS95] uses the same principles, but, as we explain later, with several differences.

A. In the first iteration the algorithm needs to decide which views are *relevant* to the query. A view is relevant if it is usable in answering the query (illustrated by the conditions in Section 4). The corresponding step in a traditional optimizer is trivial: a relation is relevant to the query if it is mentioned in the from clause.

In our example, the algorithm will determine that all three views are relevant to the query, because each of them mentions the relations in the query and applies some of the same join predicates as in the query. Therefore, the algorithm chooses the best access path to each of the views, depending on the existing index structures and selection predicates in the query.

B. Since the query execution plans involve joins over views, rather than joins over database relations, plans can no longer be neatly partitioned into equivalence classes which can be explored in increasing size. This observation implies several changes to the traditional algorithm:

1. **Termination testing:** the algorithm needs to distinguish *partial query execution plans* of the query from *complete execution plans*. The enumeration of the possible join orders terminates when there are no more unexplored partial plans. In contrast, in the traditional setting the algorithm terminates after considering the equivalence classes that include all the relations in the query.
2. **Pruning of plans:** a traditional optimizer compares between pairs of plans *within* one equivalence class and saves only the cheapest one for each class. In our context, the query optimizer needs to compare between *any pair* of plans generated thus far. A plan p is pruned if there is another plan p' that (1) is cheaper than p and, (2) has greater or equal contribution to the query than p. Informally, a plan p' contributes more to the query than the plan p if it covers more of the relations in the query and selects more of the necessary attributes.
3. **Combining partial plans:** in the traditional setting, when two partial plans are combined, the join predicates that involve both plans are explicit in the query, and the enumeration algorithm need only consider the most efficient way to apply these predicates. However, in our case, it may not be obvious a priori which join predicate will yield a correct rewriting of the query, since we are joining views rather than database relations directly. Hence, the enumeration algorithm needs to consider several alternative join predicates. Fortunately, in practice, the number of join predicates that need to be considered can be significantly pruned using meta-data about the schema. For example, there is no point in trying to join a string attribute with a numeric one. Furthermore, in some cases we can use knowledge of integrity constraints and the structure of the query to reduce the number of join predicates we consider. Finally, after considering all the possible join predicates, the optimizer also needs to check whether the resulting plan is still a partial solution to the query.

In our example, the algorithm will consider in the second iteration all possible methods to join pairs of plans produced in the first iteration. The algorithm will save the cheapest plan for each of the two-way joins, assuming the result is still a partial or complete solution to the query. The algorithm will consider the following combinations (in this discussion we ignore the choice of inner versus outer input to the join):

- the join of V1 and V2 on the attribute student: This join produces a partial result to the query. There are two ways to extend this join to complete execution plan. The first is to apply an additional selection on the c-number attribute and a projection on student and dept. The second, which is explored in the subsequent iteration, is to join the result with V3. Hence, the algorithm produces one complete execution plan and keeps V1 $\bowtie$ V2 for the subsequent iterations.
  In principle, as explained in bullet 3 above, the algorithm should also consider joining V1 and V2 on other attributes (e.g., V1.student=V2.c-number), but in this case, a simple semantic analysis shows that such a join will not yield a partial solution.
- the joins of V1 with V3 (on dept) and of V2 with V3 (on c-number): These two joins produce partial solutions to the query, but only if set semantics are considered (otherwise, the resulting rewriting will have multiple occurrences of the Major (or Registered) relation, whereas the query has only one occurrence).

In the third iteration, the algorithm tries to join the plans for the partial solutions from the second iteration with a plan from the first iteration. One of the plans the algorithm will consider is the one in which the result of joining V2 and V3 is then joined with V1. Even though this plan may seem redundant compared to V1 $\bowtie$ V2, it may be cheaper depending on the available indexes on the views, because it enables pruning the (possibly larger) set of students based on the selective course number.

Variations on the above principles are presented in [TSI94, TSI96] and [CKPS95]. The algorithm in [TSI96] attempts to reformulate a query on a logical schema to refer directly to GMAPs storing the data (see Section 2). They consider select-project-join queries with set semantics. To test whether a solution is complete (i.e., whether it is equivalent to the original query) they use an efficient and sufficient query-equivalence condition that also makes use of some inclusion and functional dependencies.

The goal of the algorithm described in [CKPS95] is to make use of materialized views in query evaluation. They consider select-project-join queries with bag semantics and which may also include arithmetic comparison predicates. Under bag semantics, the ways in which views may be combined to answer a query are more limited. This is due to the fact that two queries are equivalent if and only if there is a bi-directional 1-1 mapping between the two queries, which maps the join predicates of one query to those of the other [CV93]. Hence, if we ignore the arithmetic comparison operators, a view is usable only if it is isomorphic to a subset

Conventional optimizer	Optimizer using views
Iteration 1	**Iteration 1**
a) find all possible access paths.	a1) Find all views that are *relevant* to the query.
	a2) Distinguish between partial and complete solutions to the query.
b) Compare their cost and keep the least expensive.	b) Compare all pairs of views. If one has neither greater contribution nor a lower cost than the other, prune it.
c) If the query has one relation, stop.	c) If there are no partial solutions, stop.
Iteration 2	**Iteration 2**
For each query join:	
a) Consider joining the relevant access paths found in the previous iteration using all possible join methods.	a1) Consider joining all partial solutions found in the previous iteration using all possible equi-join methods and trying all possible subsets of join predicates.
	a2) Distinguish between complete and partial solutions.
b) Compare the cost of the resulting join plans and keep the least expensive.	b) If any newly generated solution is either not relevant to the query, or dominated by another, prune it.
c) If the query has only 2 relations, stop.	c) If there are no partial solutions, stop.
Iteration 3	**Iteration 3**
...	...

Fig. 5 A comparison of a traditional query optimizer with one that exploits materialized views.

of the query. An additional difference between [TSI96] and [CKPS95] is that the latter searches the space of join orderings in a top-down fashion, compared to the bottom-up fashion in [TSI96]. However, since the algorithms consider different semantics, their search spaces are incomparable. Both [TSI96] and [CKPS95] present experimental results that examine the cost of considering materialized views in query optimization.

5.2 Transformational and other approaches to view rewriting

In this section we describe several works that incorporate view rewriting as transformations. The common theme in these works is that replacing some part of a query with a view is considered as another transformation available to the optimizer. This approach is necessary when (1) the entire optimizer is transformational (e.g, in [GL01]), and (2) in the logical rewriting phase of a System-R style optimizer that is considering more complex SQL queries (as in [ZCL$^+$00]).

In [GL01] the authors describe an algorithm for rewriting queries using views that is implemented in the transformational optimizer of Microsoft SQL Server. In the algorithm, view matching is added as another transformation rule in the optimizer. The transformation rule is invoked on select-project-join-group-by (SPJG) expressions, and it attempts to replace the SPJG expression by a single view. The authors describe in detail the conditions under which a sub-query is replaced by a view. The key novelty in this work is the *filter-tree*, a clever index structure that makes it possible to efficiently filter the set of views that are relevant to a particular SPJG expression. The index is composed of several sub-indexes, each of which is built on a particular property of the views (e.g., the set of tables in the view, the set of output columns, grouping columns). The sub-indexes are combined in a hierarchical fashion into the filter tree, where each level in the tree further partitions the views according to another property. The authors describe a set of experiments that shows that their algorithm adds relatively little to the optimization time, even in the presence of 1000 views.

In [ZCL$^+$00] the authors describe how view rewriting is incorporated into the query rewrite phase of the IBM DB2 UDB optimizer. Their algorithm operates on the Query Graph Model (QGM) representation of a query [HFLP89], which decomposes the query into multiple QGM *boxes*, each corresponding to a select-project-join block. The algorithm attempts to match pairs of QGM boxes in the views with those in the query. The algorithm navigates the QGM in a bottom up fashion, starting from the leaf boxes. A match between a box in the query and in the view can be either (1) exact, meaning that the two boxes represent equivalent queries, or (2) may require a *compensation*. A compensation represents a set of additional operations that need to be performed on a box of the view in order to obtain an equivalent result to a box in the query. The algorithm considers a pair of boxes only after the match algorithm has been applied to every possible pair of their children. Therefore, the match (and corresponding compensation) can be determined without looking into the subtrees of their children. The algorithm terminates when it finds a match between the root of the view and some box in the QGM of the query. The authors show that by considering rewritings at the QGM level, they are able to extend previous algorithms to handle SQL queries and views with multiple blocks, while previous algorithms considered only single block queries. As we point out in the next section, their algorithm also extends previous work to handle more complex types of grouping and aggregation.

In [DPT99] the authors use a transformational approach to uniformly incorporate the use materialized views, specialized indexes and semantic integrity constraints. All of these are represented as constraints. Their procedure involves two phases, each involving a different set of transformations. In the first phase, the *chase,* the query is expanded to include any other structure (e.g,. materialized view or access structure) that is relevant to the query, resulting in a *universal* query plan. In the second phase, the *back-chase,* the optimizer tries to remove structures (and hence joins) from the universal plan, in order to obtain a plan of minimal cost. The chase procedure is based on a generalization of the standard chase procedure to handle path conjunctive queries [PT99], thereby enabling the algorithm to handle certain forms of object-oriented queries. In [PDST00] the authors describe an implementation of the framework and experiments that prove its feasibility, focusing on methods for speeding up the back-chase phase.

In [BDD$^+$98] the authors describe a limited use of transformation rules to incorporate view rewriting algorithm into the Oracle 8i DBMS. The algorithm works in two phases. In the first phase, the algorithm applies a set of rewrite rules that attempt to replace parts of the query with references to existing materialized views. The rewrite rules consider the cases in which views satisfy the conditions described in Section 4, and also consider common integrity constraints encountered in practice, such as functional dependencies and foreign key constraints. The result of the rewrite phase is a query that refers to the views. In the second phase, the algorithm compares the estimated cost of two plans: the cost of the result of the first phase, and the cost of the best plan found by the optimizer that does *not* consider the use of materialized views. The optimizer chooses to execute the cheaper of these two plans. The main advantage of this approach is its ease of implementation, since the capability of using views is added to the optimizer without changing the join enumeration module. On the other hand, the algorithm considers the cost of only one possible rewriting of the query using the views, and hence may miss the cheapest use of the materialized views.

Finally, in [ALU01] the authors consider using views for query optimization from a different angle. They consider the problem of finding the rewriting of the query with minimal cost under three specific cost models: (1) minimizing the number of views in the rewriting (hence the number of joins), (2) reducing the size of the intermediate relations computed during the rewriting, and (3) reducing the size of intermediate relations while also dropping irrelevant attributes as the computation proceeds. The techniques underlying the CoreCover algorithm described in [ALU01] are closer in spirit to those used in the MiniCon Algorithm [PL00] described in Section 6.4.

5.3 Queries with grouping and Aggregation

In decision support applications, when queries contain grouping and aggregation, there is even more of an opportunity to obtain significant speedups by reusing the results of materialized views. However, the presence of grouping and aggregation in the queries or the views introduces several new difficulties to the problem of answering queries using views. The first difficulty that arises is dealing with aggregated columns. Recall that for a view to be usable by a query, it must not project out an attribute that is needed in the query (and is not otherwise recoverable). When a view performs an aggregation on an attribute, we lose some information about the attribute, and in a sense *partially* projecting it out. If the query requires the same or a coarser grouping than performed in the view, and the aggregated column is either available or can be reconstructed from other attributes, then the view is still usable for the query. The second difficulty arises due to the loss of multiplicity of values on attributes on which grouping is performed. When we group on an attribute A, we lose the multiplicity of the attribute in the data, thereby losing the ability to perform subsequent sum, counting or averaging operations. In some cases, it may be possible to recover the multiplicity using additional information.

The following simple example illustrates some of the subtleties that arise in the presence of grouping and aggregation. To make this example slightly more appealing, we assume the quarter attribute of the relation Teaches is replaced by a year attribute (and hence, there are likely to be several offerings of the same course during an academic year). Suppose we have the following view available, which considers all the graduate level courses, and for every pair of course and year, gives the maximal course evaluation for that course in the given year, and the number of times the course was offered.

```
create view V as
select   c-number, year, Max(evaluation) as maxeval, Count(*) as offerings
from     Teaches
where    c-number ≥ 400
groupBy  c-number, year.
```

The following query considers only Ph.D-level courses, and asks for the maximal evaluation obtained for *any* course during a given year, and the number of different course offerings during that year.

```
select   year, count(*), Max(evaluation)
from     Teaches
where    c-number ≥ 500
groupBy  year.
```

The following rewriting uses the view V to answer our query.

```
select   year, sum(offerings), Max(maxeval)
from     V
where    c-number ≥ 500
groupBy  year.
```

There are a couple of points to note about the rewriting. First, even though the view performed an aggregation on the attribute evaluation, we could still use the view in the query, because the groupings in the query (on year) are more coarse than those in the view (on year and c-number). Thus, the answer to the query can be obtained by coalescing groups from the view. Second, since the view groups the answers by c-number and thereby loses the multiplicity of each course, we would have ordinarily not been able to use the view to compute the number of course offerings per year. However, since the view also computed the attribute offerings, there was still enough information in the view to recover the total number of course offerings per year, by summing the offerings per course.

Several works considered the problem of answering queries using views in the presence of grouping and aggregation. One approach considered involved a set of transformations in the query rewrite phase [GHQ95]. In this approach, the algorithm performs syntactic transformations on the query until it is possible to identify a subexpression of the query that is identical to the view, and hence substitute the view for the subexpression. However, as the authors point out, the purely syntactic nature of this approach is a limiting factor in its applicability.

A more semantic approach is proposed in [SDJL96]. The authors describe the conditions required for a view to be usable for answering a query in the presence of grouping and aggregation, and a rewriting algorithm that incorporates these conditions. That paper considers the cases in which the views and/or the queries contain grouping and aggregation. It is interesting to note that when the view contains grouping and aggregation but the query does not, then unless the query removes duplicates in the select clause, the view cannot be used to answer a query. Another important point to recall about this context is that because of the bag semantics a view will be usable to answer a query only if there is an isomorphism between the view and a subset of the query [CV93]. The work described in [ZCL$^+$00] extends the treatment of grouping and aggregation to consider multi-block queries and to multi-dimensional aggregation functions such as cube, roll-up and grouping sets [GBLP98].

Several works [CNS99, GRT99, GT00] consider the formal aspects of answering queries using views in the presence of grouping and aggregation. They present cases in which it can be shown that a rewriting algorithm is complete, in the sense that it will find a rewriting if one exists. Their algorithms are based on insights into the problem of query containment for queries with grouping and aggregation.

An interesting issue that has not received attention to date is extending the notion of maximally-contained rewritings to the presence of grouping and aggregation. In particular, one can imagine a notion of maximally-contained plans in which the answers provide the best possible *bounds* on the aggregated columns.[5]

6 Answering Queries Using Views for Data Integration

The previous section focused on extending query optimizers to accommodate the use of views. They were designed to handle cases where the number of views is relatively small (i.e., comparable to the size of the database schema), and cases where we require an equivalent rewriting of the query. In addition, for the most part, these algorithms did not consider cases in which the resulting rewriting may contain a union over the views.

In contrast, the context of data integration requires that we consider a large number of views, since each data source is being described by one or more views. In addition, the view definitions contain many complex predicates, whose goal is to express fine-grained distinctions between the contents of different data sources. As shown in Section 2, we will often not be able to find an equivalent rewriting of the query using the source views, and the best we can do is find the maximally-contained rewriting of the query. The maximally-contained rewriting will often involve a union of several queries over the sources. Furthermore, in the context of data integration it is often assumed that the views are not complete, i.e., they may only contain a subset of the tuples satisfying their definition.

In this section we describe algorithms for answering queries using views that were developed specifically for the context of data integration. These algorithms are the *bucket algorithm* developed in the context of the Information Manifold system [LRO96b] and later studied in [GM99a], the *inverse-rules algorithm* [Qia96, DGL00] which was implemented in the InfoMaster system [DG97b], and the MiniCon algorithm [PL00, PH01]. It should be noted that unlike the algorithms described in the previous section, the output of these algorithms is not a query execution plan, but rather a query referring to the view relations.

6.1 Datalog notation

For this and the next section, it is necessary to revert to datalog notation and terminology. Hence, below we provide a brief reminder of datalog notation and of conjunctive queries [Ull89, AHV95].

Conjunctive queries are able to express select-project-join queries. A conjunctive query has the form:

$$q(\bar{X}) :- r_1(\bar{X}_1),\ \ldots,\ r_n(\bar{X}_n)$$

where q, and $r_1, \ldots, r_n$ are predicate names. The predicate names $r_1, \ldots, r_n$ refer to database relations. The atom $q(\bar{X})$ is called the *head* of the query, and refers to the answer relation. The atoms $r_1(\bar{X}_1), \ldots, r_n(\bar{X}_n)$ are the *subgoals* in the body of the query. The tuples $\bar{X}, \bar{X}_1, \ldots, \bar{X}_n$ contain either variables or constants. We require that the query be *safe*, i.e., that $\bar{X} \subseteq \bar{X}_1 \cup \ldots \cup \bar{X}_n$ (that is, every variable that appears in the head must also appear in the body).

Queries may also contain subgoals whose predicates are arithmetic comparisons $<, \leq, =, \neq$. In this case, we require that if a variable X appears in a subgoal of a comparison predicate, then X must also appear in an ordinary subgoal. We refer to the subgoals of comparison predicates of a query Q by $C(Q)$.

As an example of expressing an SQL query in datalog, consider the following SQL query asking for the students (and their advisors) who took courses from their advisors after the winter of 1998:

```
select   Advises.prof, Advises.student
from     Registered, Teaches, Advises
where    Registered.c-number=Teaches.c-number and Registered.quarter=Teaches.quarter and
         Advises.prof=Teaches.prof and Advises.student=Registered.student and
         Registered.quarter > "winter98".
```

In the notation of conjunctive queries, the above query would be expressed as follows:

[5] I thank an anonymous reviewer for suggesting this problem.

q(prof, student) :-Registered(student, c-number, quarter), Teaches(prof, c-number, quarter),
Advises(prof, student), quarter > "winter98".

Note that when using conjunctive queries, join predicates of SQL are expressed by multiple occurrences of the same variable in different subgoals of the body (e.g., the variables quarter, c-number, prof, and student above). Unions can be expressed in this notation by allowing a set of conjunctive queries with the same head predicate.

A datalog query is a set of rules, each having the same form as a conjunctive query, except that predicates in the body do not have to refer to database relations. In a datalog query we distinguish EDB (extensional database) predicates that refer to the database relations from the IDB (intensional database) predicates that refer to intermediate computed relations. Hence, in the rules, EDB predicates appear only in the bodies, whereas the IDB predicates may appear anywhere. We assume that every datalog query has a distinguished IDB predicate called the *query predicate*, referring to the relation of the result.

A predicate p in a datalog program is said to *depend* on a predicate q if q appears in one of the rules whose head is p. The datalog program is said to be *recursive* if there is a cycle in the dependency graph of predicates. It is important to recall that if a datalog program is not recursive, then it can be equivalently rewritten as a union of conjunctive queries, though possibly with an exponential blowup in the size of the query. As we see in Section 7.2, certain cases may require rewritings that are recursive datalog queries.

The input to a datalog query Q consists of a database D storing extensions of all EDB predicates in Q. Given such a database D, the answer to Q, denoted by $Q(D)$, is the least fixpoint model of Q and D, which can be computed as follows. We apply the rules of the program in an arbitrary order, starting with empty extensions for the IDB relations. An application of a rule may derive new tuples for the relation denoted by the predicate in the head of the rule. We apply the rules until we cannot derive any new tuples. The output $Q(D)$ is the set of tuples computed for the query predicate. Note that since the number of tuples that can be computed for each relation is finite and monotonically increasing, the evaluation is guaranteed to terminate. Finally, we say that a datalog query refers only to views if instead of EDB predicates we have predicates referring to views (but we still allow arithmetic comparison predicates and IDB predicates).

6.2 The Bucket Algorithm

The goal of the bucket algorithm is to reformulate a user query that is posed on a mediated (virtual) schema into a query that refers directly to the available data sources. Both the query and the sources are described by conjunctive queries that may include atoms of arithmetic comparison predicates (hereafter referred to simply as predicates). As we explain in Section 7, the number of possible rewritings of the query using the views is exponential in the size of the query. Hence, the main idea underlying the bucket algorithm is that the number of query rewritings that need to be considered can be drastically reduced if we first consider each subgoal in the query in isolation, and determine which views may be relevant to each subgoal.

Given a query Q, the bucket algorithm proceeds in two steps. In the first step, the algorithm creates a bucket for each subgoal in Q that is not in $C(Q)$, containing the views (i.e., data sources) that are relevant to answering the particular subgoal. More formally, to decide whether the view V should be in the bucket of a subgoal g, we consider each of the subgoals g_1 in V and do the following:

a. check whether there is a unifier θ for g and g_1, i.e., θ is a variable mapping such that $\theta(g) = \theta(g_1)$. If there is no unifier, we proceed to the next subgoal.
b. given the unifier θ, we check whether the view and the query would be compatible after the unifier is applied. Hence, we apply $\theta_{h(V)}$ to the query and to the view, where $\theta_{h(V)}$ is the same as θ but its domain does not include the existential variables in V (since only the head variables of V are part of a rewriting). Then we check two conditions: (1) that the predicates in Q and in V are mutually satisfiable, i.e., $\theta_{h(V)}(C(Q)) \wedge \theta_{h(V)}(C(V))$ is satisfiable, and (2) that θ treats the head variables occurring in g correctly, i.e., if X is a head variable that appears in position i of the subgoal g, then the variable appearing in position i of g_1 must be a head variable of V.

If the above conditions are satisfied, then we insert the atom $\theta(head(V))$ into the bucket of g. Note that a subgoal g may unify with more than one subgoal in a view V, and in that case the bucket of g will contain multiple occurrences of V.

In the second step, the bucket algorithm finds a set of *conjunctive query rewritings*, each of them being a conjunctive query that includes one conjunct from every bucket. Each of these conjunctive query rewritings represents one way of obtaining part of the answer to Q from the views. The result of the bucket algorithm is defined to be the union of the conjunctive query rewritings (since each of the rewritings may contribute different tuples). Given a conjunction, constructed from a single element from every bucket, it is a conjunctive query rewriting if either (1) the conjunction is contained in the query Q, or (2) it is possible to add atoms of comparison predicates such that the resulting conjunction is contained in Q.

Example 2 Consider the following views

V1(student,c-number,quarter,title)	:- Registered(student,c-number,quarter), Course(c-number,title), c-number≥500, quarter≥Aut98.
V2(student,prof,c-number,quarter)	:- Registered(student,c-number,quarter), Teaches(prof,c-number,quarter)
V3(student,c-number)	:- Registered(student,c-number,quarter), quarter ≤ Aut94.
V4(prof,c-number,title,quarter)	:- Registered(student,c-number,quarter), Course(c-number,title), Teaches(prof,c-number,quarter), quarter≤Aut97.

Suppose our query is:

q(S,C,P) :- Teaches(P,C,Q), Registered(S,C,Q), Course(C,T), C≥300, Q≥Aut95.

In the first step the algorithm creates a bucket for each of the relational subgoals in the query in turn. The resulting contents of the buckets are shown in Table 2. The bucket of Teaches(P,C,Q) includes views V2 and V4, since the following mapping unifies the subgoal in the query with the corresponding Teaches subgoal in the views (thereby satisfying condition (a) above):

$\{$ P $\rightarrow$ prof, C $\rightarrow$ c-number, Q $\rightarrow$ quarter $\}$.

Note that each view head in a bucket only includes variables in the domain of the mapping. Fresh variables (primed) are used for the other head variables of the view.

The bucket of the subgoal Registered(S,C,Q) contains the views V1 and V2, since the following mapping unifies the subgoal in the query with the corresponding Registered subgoal in the views:

$\{$ S $\rightarrow$ student, C $\rightarrow$ c-number, Q $\rightarrow$ quarter $\}$.

Teaches(P,C,Q)	Registered(S,C,Q)	Course(C,T)
V2(S',P,C,Q)	V1(S,C,Q,T')	V1(S',C,Q',T)
V4(P,C,T',Q)	V2(S,P',C,Q)	V4(P',C,T,Q')

Table 2 Contents of the buckets. The primed variables are those that are not in the domain of the unifying mapping.

The view V3 is not included in the bucket of Registered(S,C,Q) because after applying the unification mapping, the predicates Q ≥ Aut95 and Q ≤ Aut94 are mutually inconsistent. The view V4 is not included in the bucket of Registered(S,C,Q) because the variable student is not in the head of V4, while S is in the head of the query.

Next, consider the bucket of the subgoal Course(C,T). The views V1 and V4 will be included in the bucket because of the mapping

$\{$ C $\rightarrow$ c-number, T $\rightarrow$ title $\}$.

In the second step of the algorithm, we combine elements from the buckets. In our example, we start with a rewriting that includes the top elements of each bucket, i.e.,

q'(S,C,P) :- V2(S',P,C,Q), V1(S,C,Q,T'), V1(S', C, Q', T).

As can be checked, this rewriting can be simplified by equating the variables S and S', and Q and Q', and then removing the third subgoal, resulting with

q'(S,C,P) :- V2(S',P,C,Q), V1(S,C,Q,T').

Another possibility that the bucket algorithm would explore is:

q'(S,C,P) :- V4(P, C, T', Q), V1(S,C,Q,T'), V4(P', C, T, Q').

However, this rewriting would be dismissed because the quarters given in V1 are disjoint from those given in V4. In this case, the views V1 and V4 are relevant to the query when they are considered in *isolation*, but, if joined, would yield the empty answer.

Finally, the algorithm would also produce the rewriting

q'(S,C,P) :- V2(S,P,C,Q), V4(P, C, T', Q).

Hence, the result of the bucket algorithm is the union of two conjunctive queries, one obtains answers by joining V1 and V2, and the other by joining V2 and V4. The reader should note that in this example, as often happens in the data integration context, the algorithm produced a *maximally-contained* rewriting of the query using the views, and not an equivalent rewriting. In fact, when the query does not contain arithmetic comparison predicates (but the view definitions still may) the bucket algorithm is guaranteed to return the maximally-contained rewriting of the query using the views. □

The strength of the bucket algorithm is that it exploits the predicates in the query to prune significantly the number of candidate conjunctive rewritings that need to be considered. Checking whether a view should belong to a bucket can be done in time polynomial in the size of the query and view definition when the predicates involved are arithmetic comparisons. Hence, if the data sources (i.e., the views) are indeed distinguished by having different comparison predicates, then the resulting buckets will be relatively small. The bucket algorithm also extends naturally to cases where the query (but not the views) is a union of conjunctive queries, and to other forms of predicates in the query such as class hierarchies [LRO96a]. Finally, the bucket algorithm also makes it possible to identify opportunities for interleaving optimization and execution in a data integration system in cases where one of the buckets contains an especially large number of views [LRO96a].

The main disadvantage of the bucket algorithm is that the Cartesian product of the buckets may still be rather large. Furthermore, in the second step the algorithm needs to perform a query containment test for every candidate rewriting. The testing problem is Π_2^p-complete,[6] though only in the size of the query and the view definition, and hence quite efficient in practice.

6.3 The Inverse-rules Algorithm

Like the bucket algorithm, the inverse-rules algorithm was also developed in the context of a data integration system [DG97b]. The key idea underlying the algorithm is to construct a set of rules that *invert* the view definitions, i.e., rules that show how to compute tuples for the database relations from tuples of the views. We illustrate inverse rules with an example. Suppose we have the following view (we omit the quarter attribute of Registered for brevity in this example):

V3(dept, c-number) :- Major(student,dept), Registered(student,c-number).

We construct one inverse rule for every subgoal in the body of the view:

Major(f_1(dept,X), dept) :- V3(dept,X)
Registered(f_1(Y, c-number), c-number) :- V3(Y,c-number)

[6] For conjunctive queries with no comparison predicates, query containment is in NP because we only need to guess a containment mapping. Here, however, we need to guess a containment mapping for every possible ordering on the variables in containing query.

Intuitively, the inverse rules have the following meaning. A tuple of the form (dept,c-number) in the extension of the view V3 is a *witness* of tuples in the relations Major and Registered. The tuple (dept,c-number) is a witness in the sense that it tells us two things:

– the relation Major contains a tuple of the form (Z, dept), for some value of Z.
– the relation Registered contains a tuple of the form (Z, c-number), for the *same* value of Z.

In order to express the information that the unknown value of Z is the same in the two atoms, we refer to it using the functional term f_1(dept,c-number). Formally, f_1 is a Skolem function (see [ABS99], Pg. 96) and therefore uninterpreted. Note that there may be several values of Z in the database that cause the tuple (dept,c-number) to be in the join of Major and Registered, but all that matters is that there exists at least one such value.

In general, we create one function symbol for every existential variable that appears in the view definitions. These function symbols are used in the heads of the inverse rules.

The rewriting of a query Q using the set of views $\mathcal{V}$ is the datalog program that includes

– the inverse rules for $\mathcal{V}$, and
– the query Q.

As shown in [DG97a,DGL00], the inverse-rules algorithm returns the maximally-contained rewriting of Q using $\mathcal{V}$. In fact, the algorithm returns the maximally contained query even if Q is an arbitrary recursive datalog program.

Example 3 Suppose a query asks for the departments in which the students of the 444 course are majoring,

q(dept) :- Major(student,dept), Registered(student, 444)

and the view V3 includes the tuples:

{ (CS, 444), (EE, 444), (CS, 333) }.

The inverse rules would compute the following tuples:

Registered: { (f_1(CS,444), CS), (f_1(EE,444), EE), (f_1(CS,333), CS) }
Major: { (f_1(CS,444),444), (f_1(EE,444),444), (f_1(CS,333),333) }

Applying the query to these extensions would yield the answers CS and EE. □

In the above example we showed how functional terms are generated as part of the evaluation of the inverse rules. However, the resulting rewriting can actually be rewritten in such a way that no functional terms appear [DG97a].

There are several interesting similarities and differences between the bucket and inverse rules algorithms that are worth noting. In particular, the step of computing buckets is similar in spirit to that of computing the inverse rules, because both of them compute the views that are relevant to single atoms of the database relations. The difference is that the bucket algorithm computes the relevant views by taking into consideration the *context* in which the atom appears in the query, while the inverse rules algorithm does not. Hence, if the predicates in a view definition entail that the view cannot provide tuples relevant to a query (because they are mutually unsatisfiable with the predicates in the query), then the view will not end up in a bucket. In contrast, the query rewriting obtained by the inverse rules algorithm may contain views that are not relevant to the query. However, the inverse rules can be computed once, and be applicable to any query. In order to remove irrelevant views from the rewriting produced by the inverse-rules algorithm we need to apply a subsequent constraint propagation phase (as in [LFS97,SR92]).

A key advantage of the inverse-rules algorithm is its conceptual simplicity and modularity. As shown in [DGL00], extending the algorithm to exploit functional dependencies on the database schema, recursive queries or the existence of access-pattern limitations can be done by adding another set of rules to the inverse rules. Furthermore, the algorithm produces the maximally-contained rewriting in time that is polynomial in the size of the query and the views. Note that the algorithm does not tell us whether the maximally-contained rewriting is equivalent to the original query, which would contradict the fact that the problem of finding an equivalent rewriting is NP-complete [LMSS95] (see Section 7).

On the other hand, using the resulting rewriting produced by the algorithm for actually evaluating queries from the views has a significant drawback, since it insists on recomputing the extensions of the database relations. In our example, evaluating the inverse rules computes tuples for Registered and Major, and the query is then evaluated over these extensions. However, by doing that, we lose the fact that the view already computed the join that the query is requesting. Hence, much of the computational advantage of exploiting the materialized view is lost.

In order to obtain a more efficient rewriting from the inverse rules, we must unfold the inverse rules and remove redundant subgoals from the unfolded rules. Unfolding the rules turns out to be similar to (but still slightly better than) the second phase of the bucket algorithm, where we consider the Cartesian product of the buckets (see [PL00] for an experimental analysis).

6.4 The MiniCon algorithm

The MiniCon algorithm [PL00,PH01] addresses the limitations of the previous algorithms. The key idea underlying the algorithm is a change of perspective: instead of building rewritings by combining rewritings for each of the query *subgoals* or the database relation, we consider how each of the *variables* in the query can interact with the available views. The result is that the second phase of the MiniCon algorithm needs to consider drastically fewer combinations of views. The following example illustrates the key idea of MiniCon. Consider the query

q(D) :- Major(S, D), Registered(S, 444, Q), Advises(P, S)

and the views:

V1(dept) :- Major(student,dept), Registered(student, 444, quarter).
V2(prof, dept, area) :- Advises(prof, student), Prof(name, area)
V3(dept, c-number) :- Major(student,dept), Registered(student, c-number, quarter),
Advises(prof, student).

The bucket algorithm considers each of the subgoals in the query in isolation, and therefore puts the view V1 into the buckets of Major(student, dept) and Registered(student, 444, quarter). However, a careful analysis reveals that V1 cannot possibly be useful in a rewriting of the query. The reason is that since the variable student is not in the head of the view, then in order for V1 to be useful, it must contain the subgoal Advises(prof,student) as well. Otherwise, the join on the variable S in the query cannot be applied using the results of V1.

The MiniCon algorithm starts out like the bucket algorithm, considering which views contain subgoals that correspond to subgoals in the query. However, once the algorithm finds a partial variable mapping from a subgoal g in the query to a subgoal g_1 in a view V, it changes perspective and looks at the variables in the query. The algorithm considers the join predicates in the query (which are specified by multiple occurrences of the same variable) and finds the minimal additional set of subgoals that *must* to be mapped to subgoals in V, given that g will be mapped to g_1. This set of subgoals and mapping information is called a *MiniCon Description* (MCD), and can be viewed as a generalized bucket. Unlike buckets, MCDs are associated with *sets* of subgoals in the query. In the second phase, the MCDs are combined to produce the query rewritings.

In the above example, the algorithm will determine that it cannot create an MCD for V1 because it cannot apply the join predicates in the query. When V2 is considered, the MCD will contain only the subgoal Advises(prof, student). When V3 is considered, the MCD will include all of the query subgoals.

The key advantage of the MiniCon algorithm is that the second phase of the algorithm considers many fewer combinations of MCDs compared to the Cartesian product of the buckets or compared to the number of unfoldings of inverse rules. The work in [PL00] describes a detailed set of experiments that shows that the MiniCon significantly outperforms the inverse rules algorithm, which in turn outperforms the bucket algorithm. The paper demonstrates exactly how these savings are obtained in the second phase of the algorithm. Furthermore, the experiments show that the algorithm scales up to hundreds of views with commonly occurring shapes such as chain, star and complete queries (see [MGA97] for a description of these query shapes). The work in [PH01] also explains how to exploit the key ideas of the the MiniCon algroithm to the context of query optimization with materialized views, where the cost of the query plan if the primary concern.

7 Theory of Answering Queries Using Views

In the previous sections we discussed specific algorithms for answering queries using views. Here we consider several fundamental issues that cut across all of the algorithms we have discussed thus far, and which have been studied from a more theoretical perspective in the literature.

The first question concerns the *completeness* of the query rewriting algorithms. That is, given a set of views and a query, will the algorithm always find a rewriting of the query using the views if one exists? A related issue is characterizing the complexity of the query rewriting problem. We discuss these issues in Section 7.1.

Completeness of a rewriting algorithm is characterized w.r.t. a specific query language in which the rewritings are expressed (e.g., select-project-join queries, queries with union, recursion). For example, there are cases in which if we do not allow unions in the rewriting of the query, then we will not be able to find an equivalent rewriting of a query using a set of views. The language that we consider for the rewriting is even more crucial when we consider maximally-contained rewritings, because the notion of maximal containment is defined w.r.t. a specific query language. As it turns out, there are several important cases in which a maximally-contained rewriting of a query can only be found if we consider *recursive* datalog rewritings. These cases are illustrated in Section 7.2.

At the limit, we would like to be able to extract all the *certain* answers for a query given a set of views, whether we do it by applying a query rewriting to the extensions of the views or via an arbitrary algorithm. In Section 7.3 we consider the complexity of finding all the certain answers, and show that even in some simple cases the problem is surprisingly co-NP-hard in the size of the extensions of the views.

7.1 Completeness and complexity of finding query rewritings

The first question one can ask about an algorithm for rewriting queries using views is whether the algorithm is complete: given a query Q and a set of views $\mathcal{V}$, will the algorithm find a rewriting of Q using $\mathcal{V}$ when one exists. The first answer to this question was given for the class of queries and views expressed as conjunctive queries [LMSS95]. In that paper it was shown that when the query does not contain comparison predicates and has n subgoals, then there exists an equivalent conjunctive rewriting of Q using $\mathcal{V}$ only if there is a rewriting with at most n subgoals. An immediate corollary of the bound on the size of the rewriting is that the problem of finding an equivalent rewriting of a query using a set of views is in NP, because it suffices to guess a rewriting and check its correctness.[7]

The bound on the size of the rewriting also sheds some light on the algorithms described in the previous sections. In particular, it entails that the search strategy that the GMAP algorithm [TSI96] employs is guaranteed to be complete under the conditions that (1) we use a sound and complete algorithm for query containment for testing equivalence of rewritings, (2) when combining two subplans, the algorithm considers all possible join predicates on the attributes of the combined subplans, and (3) we consider self-joins of the views. These conditions essentially guarantee that the algorithm searches through all rewritings whose size is bounded by the size of the query. It is important to emphasize that the rewriting of the query that produces the most *efficient* plan for answering the query may have *more* conjuncts that the original query. The bound of [LMSS95] also guarantees that the bucket algorithm is guaranteed to find the maximally-contained rewriting of the query when the query does not contain arithmetic comparison predicates (but the views may), and that we consider unions of conjunctive queries as the language for the rewriting.

In [LMSS95] it is also shown that the problem of finding a rewriting is NP-hard for two independent reasons: (1) the number of possible ways to map a single view into the query, and (2) the number of ways to combine the mappings of different views into the query.

In [RSU95] the authors extend the bound on the size of the rewriting to the case where the views contain access-pattern limitations (discussed in detail in Section 8.2). In [CR97] the authors exploit the close connection between the containment and rewriting problems, and show several polynomial-time cases of the rewriting problems, corresponding to analogous cases for the problem of query containment.

[7] Note that checking the correctness of a rewriting is NP-complete; however, the guess of a rewriting can be extended to a guess for containment mappings showing the equivalence of the rewriting and of the query.

7.2 The need for recursive rewritings

As noted earlier, in cases where we cannot find an equivalent rewriting of the query using a set of views, we consider the problem of finding maximally-contained rewritings. Our hope is that when we apply the maximally-contained rewriting to the extensions of the views, we will obtain the set of *all* certain answers to the query (Definition 4). Interestingly, there are several contexts where in order to achieve this goal we need to consider recursive datalog rewritings of the query [DGL00]. We recall that a datalog rewriting is a datalog program in which the base (EDB) predicates are the view relations, and there are additional intermediate IDB relations. Except for the obvious case in which the query is recursive [DG97a], other cases include: when we exploit the presence of functional dependencies on the database relations or when there are access-pattern limitations on the extensions of the views [DL97] (see Section 8.2 for a more detailed discussion), when views contain unions [Afr00] (though even recursion does not always suffice here), and the case where additional semantic information about class hierarchies on objects is expressed using description logics [BLR97]. We illustrate the case of functional dependencies below.

Example 4 To illustrate the need for recursive rewritings in the presence of functional dependencies, we temporarily venture into the domain of airline flights. Suppose we have the following database relation

schedule(Airline,Flight_*no*,Date,Pilot,Aircraft)

which stores tuples describing the pilot that is scheduled for a certain flight, and the aircraft that is used for this flight. Assume we have the following functional dependencies on the relations in the mediated schema

Pilot → Airline and
Aircraft → Airline

expressing the constraints that pilots work for only one airline, and that there is no joint ownership of aircrafts between airlines. Suppose we have the following view available, which projects the date, pilot and aircraft attributes from the database relation:

v(D,P,C) :- schedule(A,N,D,P,C)

The view v records on which date which pilot flies which aircraft. Now consider a query asking for pilots that work for the same airline as Mike (expressed as the following self join on the attribute Airline of the schedule relation):

q(P) :- schedule(A,N,D,'mike',C), schedule(A,N',D',P,C')

The view v doesn't record the airlines that pilots work for, and therefore, deriving answers to the above query requires using the functional dependencies in subtle ways. For example, if both Mike and Ann are known to have flown aircraft #111, then, since each aircraft belongs to a single airline, and every pilot flies for only one airline, Ann must work for the same airline as Mike. Moreover, if, in addition, Ann is known to have flown aircraft #222, and John has flown aircraft #222 then the same line of reasoning leads us to conclude that Ann and John work for the same airline. In general, for any value of n, the following conjunctive rewriting is a contained rewriting:

$$q_n(P) :- \mathsf{v}(D_1, \mathsf{mike}, C_1),\ \mathsf{v}(D_2, P_2, C_1),\ \mathsf{v}(D_3, P_2, C_2),\ \mathsf{v}(D_4, P_3, C_2),\ \ldots, \\ \mathsf{v}(D_{2n-2}, P_n, C_{n-1}),\ \mathsf{v}(D_{2n-1}, P_n, C_n),\ \mathsf{v}(D_{2n}, P, C_n)$$

Moreover, for each n, $q_n(P)$ may provide answers that were not given by q_i for $i < n$, because one can always build an extension of the view v that requires n steps of chaining in order to find answers to the query. The conclusion is that we cannot find a maximally-contained rewriting of this query using the views if we only consider non-recursive rewritings. Instead, the maximally-contained rewriting is the following datalog program:

```
relevantPilot("mike").
relevantAirCraft(C)  :- v(D, "mike", C).
relevantAirCraft(C)  :- v(D,P,C), relevantPilot(P).
relevantPilot(P)     :- relevantPilot(P1), relevantAirCraft(C), v(D1, P1, C), v(D2, P, C).
```

In the program above, the relation relevantPilot will include the set of pilots who work for the same airline as Mike, and the relation relevantAirCraft will include the aircraft flown by relevant pilots. Note that the fourth rule is mutually recursive with the definition of relevantAirCraft. □

In [DL97,DGL00] it is shown how to augment the inverse-rules algorithm to incorporate functional dependencies. The key element of that algorithm is to add a set of rules that simulate the application of a Chase algorithm [MMS79] on the atoms of the database relations.

7.3 Finding the certain answers

A different perspective on the problem of answering queries using views is the following. Given a set of materialized views and the corresponding view definitions, we obtain some *incomplete* information about the contents of the database. More specifically, the views define a set of *possible* underlying databases $\mathcal{D}$. Given a query Q over the database and a tuple t, there are a few possibilities: (1) t would be an answer to Q for every database in $\mathcal{D}$, (2) t is an answer to Q for some database in $\mathcal{D}$, or (3) t is not an answer to Q for any database in $\mathcal{D}$. The notion of certain answers, (see Definition 4) formalizes case (1).

If Q' is an *equivalent rewriting* of a query Q using the set of views $\mathcal{V}$, then it will always produce the same result as Q, independent of the state of the database or of the views. In particular, this means that Q' will always produce all the certain answers to Q for any possible database.

When Q' is a *maximally-contained rewriting* of Q using the views $\mathcal{V}$ it may produce only a subset of the answers of Q for a given state of the database. The maximality of Q' is defined only w.r.t. the other possible rewritings in a particular query language $\mathcal{L}$ that we consider for Q'. Hence, the question that remains is how to find all the certain answers, whether we do it by applying some rewritten query to the views or by some other algorithm.

The question of finding all the certain answers is considered in detail in [AD98,GM99a]. In their analysis they distinguish the case of the *open-world assumption* from that of the *closed-world assumption.* With the closed-world assumption, the extensions of the views are assumed to contain *all* the tuples that would result from applying the view definition to the database. Under the open-world assumption, the extensions of the views may be missing tuples. The open-world assumption is especially appropriate in data integration applications, where the views describe sources that may be incomplete (see [EGW97,Lev96,Dus97] for treatments of complete sources in the data integration context). The closed-world assumption is appropriate for the context of query optimization and maintaining physical data independence, where views have actually been computed from existing database relations.

Under the open-world assumption, [AD98] show that in many practical cases, finding all the certain answers can be done in polynomial time. However, the problem becomes co-NP-hard (in the size of the view extensions!) as soon as we allow union in the language for defining the views, or allow the predicate $\neq$ in the language defining the query.

Under the closed-world assumption the situation is even worse. Even when both the views and the query are defined by conjunctive queries without comparison predicates, the problem of finding all certain answers is already co-NP-hard. The following example is the crux of the proof of the co-NP-hardness result [AD98].

Example 5 The following example shows a reduction of the problem of graph 3-colorability to the problem of finding all the certain answers. Suppose the relation edge(X,Y) encodes the edges of a graph, and the relation color(X,Z) encodes which color Z is attached to the nodes of the graph. Consider the following three views:

```
V1(X) :- color(X,Y)
V2(Y) :- color(X,Y)
V3(X,Y) :- edge(X,Y)
```

where the extension of V1 is the set of nodes in a graph, the extension of V2 is the set {red, green, blue}, and the extension of V3 is the set of edges in the graph. Consider the following query:

```
q(c) :- edge(X,Y), color(X,Z), color(Y,Z)
```

In [AD98] it is shown that c is a certain answer to q if and only if the graph encoded by edge is *not* three-colorable. Hence, they show that the problem of finding all certain answers is co-NP-hard. □

The hardness of finding all the certain answers provides an interesting perspective on formalisms for data integration. Intuitively, the result entails that when we use views to describe the contents of data sources, even if we only use conjunctive queries to describe the sources, the complexity of finding all the answers to a query from the set of sources is co-NP-hard. In contrast, using a formalism in which the relations of the mediated schema are described by views over the source relations (as in [GMPQ$^+$97]), the complexity of finding all the answers is always polynomial. Hence, this result hints that the former formalism has a greater expressive power as a formalism for data integration.

It is also interesting to note the connection established in [AD98] between the problem of finding all certain answers and computation with conditional tables [IL84]. As the authors show, the partial information about the database that is available from a set of views can be encoded as a conditional table using the formalism studied in [IL84], providing a formalization to the intuition starting out this section.

The work in [GM99a] also considers the case where the views may either be incomplete, complete, or contain tuples that don't satisfy the view definition (referred to as *incorrect* tuples). It is shown that without comparison predicates in the views or the query, when either all the views are complete or all of them may contain incorrect tuples, finding all certain answers can be done in polynomial time in the size of the view extensions. In other cases, the problem is co-NP-hard. The work in [MM01] consider the query answering problem in cases where we may have bounds on the soundness and/or completeness of the views.

Finally, [MLF00] considers the problem of relative query containment, i.e., whether the set of certain answers of a query Q_1 is always contained in the set of certain answers of a query Q_2. The paper shows that for the conjunctive queries and views with no comparison predicates the problem is Π_2^p-complete, and that the problem is still decidable in the presence of access pattern limitations.

8 Extensions to the Query Language

In this section we survey the algorithms for answering queries using views in the context of several important extensions to the query languages considered thus far. We consider extensions for Object Query Language (OQL) [FRV96,Flo96,DPT99], and views with access pattern limitations [RSU95,KW96,DL97].

8.1 Object Query Language

In [FRV96,Flo96] the authors studied the problem of answering queries using views in the context of querying object-oriented databases, and have incorporated their algorithm into the Flora OQL optimizer. In object-oriented databases the correspondence between the *logical* model of the data and the *physical* model is even less direct than in relational systems. Hence, as argued in [Flo96], it is imperative for a query optimizer for object-oriented database be based on the notion of physical data independence.

Answering queries using views in the context of object-oriented systems introduces two key difficulties. First, finding rewritings often requires that we exploit some semantic information about the class hierarchy and about the attributes of classes. Second, OQL does not make a clean distinction between the `select`, `from` and `where` clauses as in SQL. `Select` clauses may contain arbitrary expressions, and the `where` clauses also allow path navigation.

The algorithm for answering queries using views described in [Flo96] operates in two phases. In the first phase the algorithm rewrites the query into a canonical form, thereby addressing the lack of distinction between the `select`, `from` and `where` clauses. As an example, in this phase, navigational expressions are removed from the `where` clause by introducing new variables and terms in the `from` clause.

In the second phase, the algorithm exploits semantic knowledge about the class hierarchy in order to find a subexpression of the query that is matched by one of the views. When such a match is found, the subexpression in the query is replaced by a reference to the view and appropriate conditions are added in order to conserve the equivalence to the query.

We illustrate the main novelties of the algorithm with the following example from [Flo96], using a French version of our university domain. Here we have the class Universities, with subclass France.Universities and the class City. The first two classes have the attributes students, PhDstudents (a sub-attribute of students), professors and adjuncts.

Example 6 Suppose we have the following view asking for students who are at least as old as their professors, and who study in universities in small cities. Below we use the notation of OQL. Note that the select clause of OQL defines the record structure of the result. Also note the use of path expressions – for example, y in x.students means that the variable y will be bound to each of the students of the object to which x will be bound.

```
create view V1 as
select      distinct [A:=x.name, B:=y.identifier, C:=z]
from        x in Universities, y in x.students, z in union(x.professors, x.adjuncts)
where       x.city.kind="small" and y.age ≥ z.age.
```

Suppose a query asks for Ph.D students in French universities who have the same age as their professors, and study in small town universities:

```
select      distinct [D:=u.name, E:=v.name, F:=t.name]
from        u in France.Universities, v in u.PhDstudents, t in u.professors
where       u.city.kind="small" and v.age=t.age.
```

In the first step, the algorithm will transform the query and the view into their normal form. The resulting expression for the query would be: (note that the variable w was added to the query in order to eliminate the navigation term from the where clause)

```
select     distinct [D:=u.name, E:=v.name, F:=t.name]
from       u in France.Universities, w in City, v in u.PhDstudents, t in u.professors
where      w.kind="small" and v.age=t.age and u.city=w.
```

In the next step, the algorithm will note the following properties of the schema:

1. The collection France.Universities is included in the collection Universities,
2. The collection denoted by the expression u.PhDstudents is included in the collection denoted by x.students. This inclusion follows from the first inclusion and the fact that PhD students are a subset of students.
3. The collection u.professors is included in the collection union(x.professors, x.adjuncts).

Putting these three inclusions together, the algorithm determines that the view can be used to answer the query, because the selections in the view are less restrictive than those in the query. The rewriting of the query using the view is the following:

```
select     distinct [D:=a.A, E:=a.B.name, F:=t.name]
from       a in V1, u in France.Universities, v in u.PhDstudents, t in u.professors
where      u.city.kind="small" and v.age=t.age and
           u.name=a.A and v.name=a.B and t=a.C.
```

Note that the role of the view is only to restrict the possible bindings of the variables used in the query. In particular, the query still has to restrict the universities to only the French ones, the students to only the Ph.Ds, and the range of the variable t to cover only professors. In this case, the evaluation of the query using the view is likely to be more efficient than computing the query only from the class extents. □

As noted in Section 5.2, the algorithm described in [DPT99,PDST00] also considers certain types of queries over object-oriented data.

8.2 Access Pattern Limitations

In the context of data integration, where data sources are modeled as views, we may have limitations on the possible access paths to the data. For example, when querying the Internet Movie Database, we cannot simply ask for all the tuples in the database. Instead, we must supply one of several inputs, (e.g., actor name or director), and obtain the set of movies in which they are involved.

We can model limited access paths by attaching a set of adornments to every data source. If a source is modeled by a view with n attributes, then an adornment consists of a string of length n, composed of the letters b and f. The meaning of the letter b in an adornment is that the source *must* be given values for the attribute in that position. The meaning of the letter f in an adornment is that the source doesn't have to be given a value for the attribute in that position. For example, an adornment bf for a view $V(A, B)$ means that tuples of V can be obtained only by providing values for the attributes A.

Several works have considered the problem of answering queries using views when the views are also associated with adornments describing limited access patterns. In [RSU95] it is shown that the bound given in [LMSS95] on the length of a possible rewriting does not hold anymore. To illustrate, consider the following example, where the binary relation Cites stores pairs of papers X, Y, where X cites Y. Suppose we have the following views with their associated adornments:

CitationDB^{bf}(X,Y) :- Cites(X,Y)
CitingPapers^{f}(X) :- Cites(X,Y)

and suppose we have the following query asking for all the papers citing paper #001:

Q(X) :- Cites(X,001)

The bound given in [LMSS95] would require that if there exists a rewriting, then there is one with at most one atom, the size of the query. However, the only possible rewriting in this case is:

q(X) :- CitingPapers(X), CitationDB(X,001).

[RSU95] shows that in the presence of access-pattern limitations it is sufficient to consider a slightly larger bound on the size of the rewriting: $n + v$, where n is the number of subgoals in the query and v is the number of variables in the query. Hence, the problem of finding an equivalent rewriting of the query using a set of views is still NP-complete.

The situation becomes more complicated when we consider maximally-contained rewritings. As the following example given in [KW96] shows, there may be *no* bound on the size of a rewriting. In the following example, the relation DBpapers denotes the set of papers in the database field, and the relation AwardPapers stores papers that have received awards (in databases or any other field). The following views are available:

DBSource^{f}(X) :- DBpapers(X)
CitationDB^{bf}(X,Y) :- Cites(X,Y)
AwardDB^{b}(X) :- AwardPaper(X)

The first source provides all the papers in databases, and has no access-pattern limitations. The second source, when given a paper, will return all the papers that are cited by it. The third source, when given a paper, returns whether the paper is an award winner or not.

The query asks for all the papers that won awards:

Q(X) :- AwardPaper(X).

Since the view AwardDB requires its input to be bound, we cannot query it directly. One way to get solutions to the query is to obtain the set of all database papers from the view DBSource, and perform a dependent join with the view AwardDB. Another way would be to begin by retrieving the papers in DBSource, join the result with the view CitationDB to obtain all papers cited by papers in DBSource, and then join the result with the view AwardDB. As the rewritings below show, we can follow any length of citation chains beginning with papers in DBSource and obtain answers to the query that were possibly not obtained by shorter chains. Hence, there is no bound on the length of a rewriting of the query using the views.

Q'(X) :- DBSource(X), AwardDB(X)
Q'(X) :- DBSource(V), CitationDB(V,X_1), $\ldots$, CitationDB(X_n,X), AwardDB(X).

Fortunately, as shown in [DL97,DGL00], we can still find a finite rewriting of the query using the views, albeit a recursive one. The following datalog rewriting will obtain all the possible answers from the above views. The key in constructing the program is to define a new intermediate relation papers whose extension is the set of all papers reachable by citation chains from papers in databases, and is defined by a transitive closure over the view CitationDB.

```
papers(X) :- DBsource(X)
papers(X) :- papers(Y), CitationDB(Y,X)
Q'(X) :- papers(X), AwardDB(X).
```

In [DL97] it is shown that a maximally-contained rewriting of the query using the views can always be obtained with a recursive rewriting. In [FW97] and [LKG99] the authors describe additional optimizations to this basic algorithm.

8.3 Other Extensions

Several authors have considered additional extensions of the query rewriting problems in various contexts. We mention some of them here.

Extensions to the query and schema language: In [AGK99,Dus98] the authors consider the rewriting problem when the views may contain unions. The consideration of inclusion dependencies on the database relations introduces several subtleties to the query rewriting problem, which are considered in [Gry98]. In [Mil98], the author considers the query rewriting problem for a language that enables querying the schema and data uniformly, and hence, names of attributes in the data may become constants in the extensions of the views. In [MRP99] the authors show that when the schema contains a single universal relation, answering queries using views and several related operations can be done more efficiently.

Semi-structured data: The emergence of XML as a standard for sharing data on the WWW has spurred significant interest in building systems for integrating XML data from multiple sources. The emerging formalisms for modeling XML data are variations on labeled directed graphs, which have also been used to model semi-structured data [Abi97,Bun97,ABS99]. The model of labeled directed graphs is especially well suited for modeling the irregularity and the lack of schema which are inherent in XML data. Several languages have been developed for querying semi-structured data and XML [AQM$^+$97,FFLS97,BDHS96,DFF$^+$99,CRF00].

Several works have started considering the problem of answering queries using views when the views and queries are expressed in a language for querying semi-structured data. There are two main difficulties that arise in this context. First, such query languages enable using *regular path expressions* in the query, to express navigational queries over data whose structure is not well known a priori. Regular path expressions essentially provide a very limited kind of recursion in the query language. In [CGLV99] the authors consider the problem of rewriting a regular path query using a set of regular path views, and show that the problem is in 2EXPTIME (and checking whether the rewriting is an equivalent one is in 2EXPSPACE). In [CGLV00a] the authors consider the problem of finding all the certain answers when queries and views are expressed using regular path expressions, and show that the problem is co-NP-complete when data complexity (i.e., size of the view extensions) is considered. In [CGLV00b] the authors extend the results of [CGLV99,CGLV00a] to path expressions that include the inverse operator, allowing both forward and backward traversals in a graph.

The second problem that arises in the context of semi-structured data stems from the rich restructuring capabilities which enable the creation of arbitrary graphs in the output. The output graphs can also include nodes that did not exist in the input data. In [PV99] the authors consider the rewriting problem in the case where the query can create *answer trees*, and queries that do not involve regular path expressions with recursion. For the most part, considering queries with restructuring remains an open research problem.

Infinite number of views: Two works have considered the problem of answering queries using views in the presence of an *infinite* number of views [LRU96,VP97]. The motivation for this seemingly curious problem is that when a data source has the capability to perform *local* processing, it can be modeled by the (possibly infinite) set of views it can supply, rather than a single one. As a simple example, consider a data source that stores a set of documents, and can answer queries of the form:

```
q(doc) :- document(doc), contains(doc, w1), ..., contains(doc,wn)
```

where we can have any number of occurrences of the contains subgoal, each with a different word.

To answer queries using such sources, one need not only choose which sources to query, but we must also choose which query to send to it out of the set of possible queries it can answer. In [LRU96, VP97] it is shown that in certain important cases the problem of answering a query using an infinite set of views is decidable. Of particular note is the case in which the set of views that a source can answer is described by the finite unfoldings of a datalog program.

Description Logics: Description logics are a family of logics for modeling complex hierarchical structures. A description logic makes it possible to define sets of objects by specifying their properties, and then to reason about the relationship between these sets (e.g., subsumption, disjointness). A description logic also enables reasoning about individual objects, and their membership in different sets. One of the reasons that description logics are useful in data management is their ability to describe complex models of a domain and reason about inter-schema relationships [CL93]. For that reason, description logics have been used in several data integration systems [AKS96,LRO96a]. Borgida [Bor95] surveys the use of description logics in data management.

Several works have considered the problem of answering queries using views when description logics are used to model the domain. In [BLR97] it is shown that in general, answering queries using views in this context may be NP-hard, and presents cases in which we can obtain a maximally-contained rewriting of a query in recursive datalog. The complexity of answering queries using views for an expressive description logic (which also includes n-ary relations) is studied in [CGL99].

9 Conclusions

As this survey has shown, the problem of answering queries using views raises a multitude of challenges, ranging from theoretical foundations to considerations of a more practical nature. While algorithms for answering queries using views are already being incorporated into commercial database systems (e.g., [BDD$^+$98, ZCL$^+$00]), these algorithms will have even more importance in data integration systems and data warehouse design. Furthermore, answering queries using views is a key technique to give database systems the ability of maintaining physical data independence.

There are many issues that remain open in this realm. Although we have touched upon several query languages and extensions thereof, many cases remain to be investigated. Of particular note are studying rewriting algorithms in the presence of a wider class of integrity constraints on both the database and view relations, and studying the effect of restructuring capabilities of query languages (as in OQL or languages for querying semistructured data [BDHS96,AQM$^+$97,FFLS97,DFF$^+$99,CRF00]).

As described in the article, different motivations have led to two strands of work on answering queries using views, one in the context of optimization and the other in the context of data integration. In part, these differences are due to the fact that in the data integration context the algorithms search for a maximally-contained rewriting of the query and assume that the number of views is relatively large. However, as we illustrated, the principles underlying the two strands are similar. Furthermore, interesting challenges arise as we try to bridge the gaps between these bodies of work. The first challenge is to extend the work on query optimization to handle a much larger number of more complex views. The second challenge is to extend data integration algorithms to choose judiciously the best rewritings of the query. This can be done by either trying to order the access to the data sources (as in [FKL97,DL99,NLF99]), or to combine the choice of rewritings with other adaptive aspects of query processing explored in data integration systems (e.g., [UFA98,IFF$^+$99]).

The context of data warehouse design, when one tries to select a set of views to materialize in the warehouse, raises another challenge. The data warehouse design problem is often treated as a problem of *search* through a set of warehouse configurations. In each configuration, we need to determine whether the workload queries anticipated on the warehouse can be answered using the selected views, and estimate the cost of the configuration. In this context it is important to be able to reuse the results of the computation from the previous state in the search space. In particular, this raises the challenge of developing *incremental* algorithms for answering queries using views, which can compute rewritings more efficiently when only minor changes are made to the set of available views.

In this survey we considered the problem of using materialized views when they are available. I believe that the next challenge is *selecting* which views to materialize in the first place. The problem of view selection also has a surprising number of potential applications, such as query optimization, data warehousing, web-site design, content distribution networks, peer-to-peer computing and ubiquitous computing. Even though there has been work on this problem (e.g., [CHS01, ACN00, Gup97a, CG00, GM99c, TS97, YKL97, BPT97, GHRU97, HRU96, GHI+01]), the research is still in its infancy. The wealth of techniques developed for answering queries using views will be very useful in this realm.

Acknowledgments

I would like to thank Phil Bernstein, Mike Carey, Anhai Doan, Todd Millstein, Rachel Pottinger, Arnie Rosenthal, Igor Tatarinov and the anonymous reviewers for valuable comments on earlier drafts of this paper. I would like to acknowledge the support of a Sloan Fellowship, NSF Grants #IIS-9978567 and #IIS-9985114, and gifts from Microsoft Corporation and from NEC, Japan.

References

[Abi97] Serge Abiteboul. Querying semi-structured data. In *Proc. of ICDT*, pages 1–18, Delphi, Greece, 1997.

[ABS99] Serge Abiteboul, Peter Buneman, and Dan Suciu. *Data on the Web.* Morgan Kaufmann, 1999.

[ACN00] Sanjay Agrawal, Surajit Chaudhuri, and Vivek Narasayya. Automated selection of materialized views and indexes in Microsoft SQL Server. In *Proc. of VLDB*, pages 496–505, Cairo, Egypt, 2000.

[ACPS96] S. Adali, K. Candan, Y. Papakonstantinou, and V.S. Subrahmanian. Query caching and optimization in distributed mediator systems. In *Proc. of SIGMOD*, pages 137–148, Montreal, Canada, 1996.

[AD98] S. Abiteboul and O. Duschka. Complexity of answering queries using materialized views. In *Proc. of PODS*, pages 254–263, Seattle, WA, 1998.

[Afr00] Foto Afrati. Personal communication, 2000.

[AGK99] Foto Afrati, M. Gergatsoulis, and Th. Kavalieros. Answering queries using materialized views with disjunctions. In *Proc. of ICDT*, pages 435–452, 1999.

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases.* Addison Weseley, 1995.

[AKS96] Yigal Arens, Craig A. Knoblock, and Wei-Min Shen. Query reformulation for dynamic information integration. *International Journal on Intelligent and Cooperative Information Systems*, (6) 2/3:99–130, June 1996.

[ALU01] Foto Afrati, Chen Li, and Jeffrey Ullman. Generating efficient plans for queries using views. In *Proc. of SIGMOD*, pages 319–330, 2001.

[AQM+97] Serge Abiteboul, Dallan Quass, Jason McHugh, Jennifer Widom, and Janet Wiener. The Lorel query language for semistructured data. *International Journal on Digital Libraries*, 1(1):68–88, April 1997.

[BDD+98] Randall Bello, Karl Dias, Alan Downing, James Feenan, Jim Finnerty, William Norcott, Harry Sun, Andrew Witkowski, and Mohamed Ziauddin. Materialized views in Oracle. In *Proc. of VLDB*, pages 659–664, 1998.

[BDHS96] Peter Buneman, Susan Davidson, Gerd Hillebrand, and Dan Suciu. A query language and optimization techniques for unstructured data. In *Proc. of SIGMOD*, pages 505–516, Montreal, Canada, 1996.

[BLR97] Catriel Beeri, Alon Y. Levy, and Marie-Christine Rousset. Rewriting queries using views in description logics. In *Proc. of PODS*, pages 99–108, Tucson, Arizona., 1997.

[Bor95] Alex Borgida. Description logics in data management. *IEEE Trans. on Know. and Data Engineering*, 7(5):671–682, 1995.

[BPT97] Elena Baralis, Stefano Paraboschi, and Ernest Teniente. Materialized views selection in a multidimensional database. In *Proc. of VLDB*, pages 156–165, 1997.

[Bun97] Peter Buneman. Semistructured data. In *Proc. of PODS*, pages 117–121, Tucson, Arizona, 1997.

[CG00] Rada Chirkova and Michael Genesereth. Linearly bounded reformulations of conjunctive databases. In *Proc. of DOOD*, pages 987–1001, 2000.

[CGL99] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Answering queries using views in description logics. In *Working notes of the KRDB Workshop*, 1999.

[CGLV99] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Vardi. Rewriting of regular expressions and regular path queries. In *Proc. of PODS*, pages 194–204, 1999.

[CGLV00a] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Vardi. Answering regular path queries using views. In *Proc. of ICDE*, pages 389–398, 2000.

[CGLV00b] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Vardi. View-based query processing for regular path queries with inverse. In *Proc. of PODS*, pages 58–66, 2000.

[CHS01] Rada Chirkova, Alon Halevy, and Dan Suciu. A formal perspective on the view selection problem. In *Proc. of VLDB*, 2001.

[CKPS95] Surajit Chaudhuri, Ravi Krishnamurthy, Spyros Potamianos, and Kyuseok Shim. Optimizing queries with materialized views. In *Proc. of ICDE*, pages 190–200, Taipei, Taiwan, 1995.

[CL93] T. Catarci and M. Lenzerini. Representing and using interschema knowledge in cooperative information systems. *Journal of Intelligent and Cooperative Information Systems*, pages 55–62, 1993.

[CM77] A.K. Chandra and P.M. Merlin. Optimal implementation of conjunctive queries in relational databases. In *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing*, pages 77–90, 1977.

[CN98a] S. Chaudhuri and V. R. Narasayya. Autoadmin 'what-if' index analysis utility. In *Proc. of SIGMOD*, pages 367–378, 1998.

[CN98b] S. Chaudhuri and V. R. Narasayya. Microsoft index tuning wizard forSQL Server 7.0. In *Proc. of SIGMOD*, pages 553–554, 1998.

[CNS99] S. Cohen, W. Nutt, and A. Serebrenik. Rewriting aggregate queries using views. In *Proc. of PODS*, pages 155–166, 1999.

[CR94] C. Chen and N. Roussopoulos. Implementation and performance evaluation of the ADMS query optimizer. In *Proc. of EDBT*, pages 323–336, March 1994.

[CR97] C. Chekuri and A. Rajaraman. Conjunctive query containment revisited. In *Proc. of ICDT*, pages 56–70, Delphi, Greece, 1997.

[CRF00] Donald D. Chamberlin, Jonathan Robie, and Daniela Florescu. Quilt: An XML query language for heterogeneous data sources. In *WebDB (Informal Proceedings) 2000*, pages 53–62, 2000.

[CV92] Surajit Chaudhuri and Moshe Vardi. On the equivalence of recursive and nonrecursive datalog programs. In *Proc. of PODS*, pages 55–66, San Diego, CA., 1992.

[CV93] Surajit Chaudhuri and Moshe Vardi. Optimizing real conjunctive queries. In *Proc. of PODS*, pages 59–70, Washington D.C., 1993.

[CV94] Surajit Chaudhuri and Moshe Vardi. On the complexity of equivalence between recursive and nonrecursive datalog programs. In *Proc. of PODS*, pages 55–66, Minneapolis, Minnesota, 1994.

[DFF$^+$99] Alin Deutsch, Mary Fernandez, Daniela Florescu, Alon Levy, and Dan Suciu. A query language for XML. In *Proceedings of the World-Wide Web 8 Conference*, pages 1155–1169, 1999.

[DFJ$^+$96] Shaul Dar, Michael J. Franklin, Bjorn Jonsson, Divesh Srivastava, and Michael Tan. Semantic data caching and replacement. In *Proc. of VLDB*, pages 330–341, 1996.

[DFS99] Alin Deutsch, Mary Fernandez, and Dan Suciu. Storing semi-structured data with STORED. In *Proc. of SIGMOD*, pages 431–442, 1999.

[DG97a] Oliver M. Duschka and Michael R. Genesereth. Answering recursive queries using views. In *Proc. of PODS*, pages 109–116, Tucson, Arizona., 1997.

[DG97b] Oliver M. Duschka and Michael R. Genesereth. Query planning in infomaster. In *Proceedings of the ACM Symposium on Applied Computing*, pages 109–111, San Jose, CA, 1997.

[DGL00] Oliver Duschka, Michael Genesereth, and Alon Levy. Recursive query plans for data integration. *Journal of Logic Programming, special issue on Logic Based Heterogeneous Information Systems*, 43(1):49–73, 2000.

[DL97] Oliver M. Duschka and Alon Y. Levy. Recursive plans for information gathering. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 778–784, 1997.

[DL99] Anhai Doan and Alon Levy. Efficiently ordering query plans for data integration. In *IJCAI Workshop on Intelligent Data Integration*, Stockholm, Sweden, August 1999.

[DPT99] Alin Deutsch, Lucian Popa, and Val Tannen. Physical data independence, constraints and optimization with universal plans. In *Proc. of VLDB*, pages 459–470, 1999.

[Dus97] Oliver Duschka. Query optimization using local completeness. In *Proceedings of the AAAI Fourteenth National Conference on Artificial Intelligence*, pages 249–255, 1997.

[Dus98] Oliver M. Duschka. *Query Planning and Optimization in Information Integration.* PhD thesis, Stanford University, Stanford, California, 1998.

[EGW97] Oren Etzioni, Keith Golden, and Daniel S. Weld. Sound and efficient closed-world reasoning for planning. *Artificial Intelligence*, 89(1-2):113–148, 1997.

[FFLS97] Mary Fernandez, Daniela Florescu, Alon Levy, and Dan Suciu. A query language for a web-site management system. *SIGMOD Record*, 26(3):4–11, September 1997.

[FK99] D. Florescu and D. Kossmann. Storing and querying XML data using an rdbms. *IEEE Data Engeneering Bulletin*, 22(3):27–34, September 1999.

[FKL97] Daniela Florescu, Daphne Koller, and Alon Levy. Using probabilistic information in data integration. In *Proc. of VLDB*, pages 216–225, Athens, Greece, 1997.

[FLM98] Daniela Florescu, Alon Levy, and Alberto Mendelzon. Database techniques for the world-wide web: A survey. *SIGMOD Record*, 27(3):59–74, September 1998.

[Flo96] Daniela D. Florescu. *Search Spaces for Object-Oriented Query Optimization.* PhD thesis, Univerisity of Paris VI, France, 1996.

[FLSY99] Daniela Florescu, Alon Levy, Dan Suciu, and Khaled Yagoub. Optimization of run-time management of data intensive web sites. In *Proc. of VLDB*, pages 627–638, 1999.

[FRV96] Daniela Florescu, Louiqa Raschid, and Patrick Valduriez. Answering queries using OQL view expressions. In *Workshop on Materialized Views, in cooperation with ACM SIGMOD*, Montreal, Canada, 1996.

[FW97] M. Friedman and D. Weld. Efficient execution of information gathering plans. In *Proceedings of the International Joint Conference on Artificial Intelligence, Nagoya, Japan*, pages 785–791, 1997.

[GBLP98] Jim Gray, Adam Bosworth, Andrew Layman, and Hamid Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab and sub-totals. In *Proc. of ICDE*, pages 152–159, 1998.

[GHI$^+$01] Steven Gribble, Alon Halevy, Zachary Ives, Maya Rodrig, and Dan Suiu. What can databases do for peer-to-peer? In *ACM SIGMOD WebDB Workshop 2001*, 2001.

[GHQ95] Ashish Gupta, Venky Harinarayan, and Dallan Quass. Aggregate-query processing in data warehousing environments. In *Proc. of VLDB*, pages 358–369, 1995.

[GHRU97] H. Gupta, V. Harinarayan, A. Rajaraman, and J. D. Ullman. Index selection for OLAP. In *Proc. of ICDE*, pages 208–219, 1997.

[GL01] Jonathan Goldstein and Per-Ake Larson. Optimizing queries using materialized views: a practical, scalable solution. In *Proc. of SIGMOD*, pages 331–342, 2001.

[GM99a] Gosta Grahne and Alberto O. Mendelzon. Tableau techniques for querying information sources through global schemas. In *Proc. of ICDT*, pages 332–347, 1999.

[GM99b] Ashish Gupta and Inderpal Mumick, editors. *Materialized Views: Techniques, Implementations and Applications.* The MIT Press, 1999.

[GM99c] H. Gupta and I. S. Mumick. Selection of views to materialize under a maintenance cost constraint. In *Proc. of ICDT*, pages 453–470, 1999.

[GMPQ$^+$97] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, and J. Widom. The TSIMMIS project: Integration of heterogeneous information sources. *Journal of Intelligent Information Systems*, 8(2):117–132, March 1997.

[GRT99] Stephane Grumbach, Maurizio Rafanelli, and Leonardo Tininini. Querying aggregate data. In *Proc. of PODS*, pages 174–184, 1999.

[Gry98] Jarek Gryz. Query folding with inclusion dependencies. In *Proc. of ICDE*, pages 126–133, Orlando, Florida, 1998.

[GT00] Stephane Grumbach and Leonardo Tininini. On the content of materialzed aggregate views. In *Proc. of PODS*, 2000.

[Gup97a] H. Gupta. Selection of views to materialize in a data warehouse. In *Proc. of ICDT*, pages 98–112, 1997.

[Gup97b] Himanshu Gupta. Selection of views to materialize in a data warehouse. In *Proc. of ICDT*, pages 98–112, Delphi, Greece, 1997.

[HFLP89] Laura Haas, Johann Freytag, Guy Lohman, and Hamid Pirahesh. Extensible query processing in Starburst. In *Proc. of SIGMOD*, pages 377–388, 1989.

[HRU96] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *Proc. of SIGMOD*, pages 205–216, 1996.

[IFF$^+$99] Zachary Ives, Daniela Florescu, Marc Friedman, Alon Levy, and Dan Weld. An adaptive query execution engine for data integration. In *Proc. of SIGMOD*, pages 299–310, 1999.

[IL84] T. Imielinski and W. Lipski. Incomplete information in relational databases. *Journal of the ACM*, 31(4):761–791, 1984.

[KB96] A. M. Keller and J. Basu. A predicate-based caching scheme for client-server database architectures. *VLDB Journal*, 5(1):35–47, 1996.

[Klu88] A. Klug. On conjunctive queries containing inequalities. *Journal of the ACM*, pages 35(1): 146–160, 1988.

[KMT98] Phokion Kolaitis, David Martin, and Madhukar Thakur. On the complexity of the containment problem for conjunctive queries with built-in predicates. In *Proc. of PODS*, pages 197–204, Seattle, WA, 1998.

[KW96] Chung T. Kwok and Daniel S. Weld. Planning to gather information. In *Proceedings of the AAAI Thirteenth National Conference on Artificial Intelligence*, pages 32–39, 1996.

[Lev96] Alon Y. Levy. Obtaining complete answers from incomplete databases. In *Proc. of VLDB*, pages 402–412, Bombay, India, 1996.

[Lev00] Alon Y. Levy. Logic-based techniques in data integration. In Jack Minker, editor, *Logic-Based Artificial Intelligence*, pages 575–595. Kluwer Academic Publishers, Dordrecht, 2000.

[LFS97] Alon Y. Levy, Richard E. Fikes, and Shuky Sagiv. Speeding up inferences using relevance reasoning: A formalism and algorithms. *Artificial Intelligence*, 97(1-2), 1997.

[LKG99] Eric Lambrecht, Subbarao Kambhampati, and Senthil Gnanaprakasam. Optimizing recursive information gathering plans. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 1204–1211, 1999.

[LMSS95] Alon Y. Levy, Alberto O. Mendelzon, Yehoshua Sagiv, and Divesh Srivastava. Answering queries using views. In *Proc. of PODS*, pages 95–104, San Jose, CA, 1995.

[LRO96a] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Query answering algorithms for information agents. In *Proceedings of the National Conference on Artificial Intelligence*, pages 40–47, 1996.

[LRO96b] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proc. of VLDB*, pages 251–262, Bombay, India, 1996.

[LRU96] Alon Y. Levy, Anand Rajaraman, and Jeffrey D. Ullman. Answering queries using limited external processors. In *Proc. of PODS*, pages 227–237, Montreal, Canada, 1996.

[LS93] Alon Y. Levy and Yehoshua Sagiv. Queries independent of updates. In *Proc. of VLDB*, pages 171–181, Dublin, Ireland, 1993.

[MGA97] M.Steinbrunn, G.Moerkotte, and A.Kemper. Heuristic and randomized optimization for the join. *VLDB Journal*, 6(3):191–208, 1997.

[Mil98] R. J. Miller. Using schematically heterogeneous structures. In *Proc. of SIGMOD*, pages 189–200, Seattle, WA, 1998.

[MLF00] Todd Millstein, Alon Levy, and Marc Friedman. Query containment for data integration systems. In *Proc. of PODS*, pages 67–75, Dallas, Texas, 2000.

[MM01] Alberto Mendelzon and George Mihaila. Querying partially sound and complete data sources. In *Proc. of PODS*, pages 162–170, 2001.

[MMS79] David Maier, Alberto Mendelzon, and Yehoshua Sagiv. Testing implications of data dependencies. *ACM Transactions on Database Systems*, 4(4):455–469, 1979.

[MRP99] Michael Minock, Marek Rusinkiewicz, and Brad Perry. The identification of missing information resources through the query difference operator. In *Proceedings of the Fourth IFCIS International Conference on Cooperative Information Systems (CoopIS 99)*, September 1999.

[NLF99] Felix Naumann, Ulf Leser, and Johann C. Freytag. Quality-driven integration of heterogeneous information systems. In *25th Conference on Very Large Database Systems (VLDB)*, pages 447–458, 1999.

[PDST00] Lucian Popa, Alin Deutsch, Arnaud Sahuguet, and Val Tannen. A chase too far? In *Proc. of SIGMOD*, pages 273–284, 2000.

[PH01] Rachel Pottinger and Alon Halevy. Minicon: A scalable algorithm for answering queries using views. *VLDB Journal*, 2001.

[PL00] Rachel Pottinger and Alon Levy. A scalable algorithm for answering queries using views. In *Proc. of VLDB*, pages 484–495, Cairo, Egypt, 2000.

[PT99] Lucian Popa and Val Tannen. An equational chase for path conjunctive queries, constraints and views. In *Proc. of ICDT*, 1999.

[PV99] Yannis Papakonstantinou and Vasilis Vassalos. Query rewriting for semi-structured data. In *Proc. of SIGMOD*, pages 455–466, 1999.

[Qia96] Xiaolei Qian. Query folding. In *Proc. of ICDE*, pages 48–55, New Orleans, LA, 1996.

[RSU95] Anand Rajaraman, Yehoshua Sagiv, and Jeffrey D. Ullman. Answering queries using templates with binding patterns. In *Proc. of PODS*, pages 105–112, San Jose, CA, 1995.

[SAC$^+$79] P. Selinger, M Astrahan, D. Chamberlin, R. Lorie, and T. Price. Access path selection in relational database systems. In *Proc. of SIGMOD*, pages 23–34, Boston, Massachusetts, 1979.

[Sag88] Yehoshua Sagiv. Optimizing datalog programs. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 659–698. Morgan Kaufmann, Los Altos, CA, 1988.

[SDJL96] Divesh Srivastava, Shaul Dar, H. V. Jagadish, and Alon Y. Levy. Answering SQL queries using materialized views. In *Proc. of VLDB*, Bombay, India, 1996.

[SGT$^+$99] J. Shanmugasundaram, H. Gang, K. Tufte, C. Zhang, D. J. DeWitt, and J. Naughton. Relational databases for querying XML documents: Limitations and opportunities. In *Proc. of VLDB*, pages 302–314, 1999.

[Shm93] Oded Shmueli. Equivalence of datalog queries is undecidable. *Journal of Logic Programming*, 15:231–241, 1993.

[SR92] Divesh Srivastava and Raghu Ramakrishnan. Pushing constraint selections. In *Proc. of PODS*, pages 301–315, San Diego, CA., 1992.

[SY81] Y. Sagiv and M. Yannakakis. Equivalence among relational expressions with the union and difference operators. *Journal of the ACM*, 27(4):633–655, 1981.

[TIHW01] Igor Tatarinov, Zachary Ives, Alon Halevy, and Dan Weld. Updating XML. In *Proc. of SIGMOD*, pages 413–424, 2001.

[TS97] Dimitri Theodoratos and Timos Sellis. Data warehouse configuration. In *Proc. of VLDB*, pages 126–135, Athens, Greece, 1997.

[TSI94] Odysseas G. Tsatalos, Marvin H. Solomon, and Yannis E. Ioannidis. The GMAP: A versatile tool for physical data independence. In *Proc. of VLDB*, pages 367–378, Santiago, Chile, 1994.

[TSI96] Odysseas G. Tsatalos, Marvin H. Solomon, and Yannis E. Ioannidis. The GMAP: A versatile tool for physical data independence. *VLDB Journal*, 5(2):101–118, 1996.

[UFA98] Tolga Urhan, Michael J. Franklin, and Laurent Amsaleg. Cost based query scrambling for initial delays. In *Proc. of SIGMOD*, pages 130–141, Seattle, WA, 1998.

[Ull89] Jeffrey D. Ullman. *Principles of Database and Knowledge-base Systems, Volumes I, II.* Computer Science Press, Rockville MD, 1989.

[Ull97] Jeffrey D. Ullman. Information integration using logical views. In *Proc. of ICDT*, pages 19–40, Delphi, Greece, 1997.

[Val87] Patrick Valduriez. Join indices. *ACM Transactions on Database Systems*, 12(2):218–246, 1987.

[VP97] Vasilis Vassalos and Yannis Papakonstantinou. Describing and using query capabilities of heterogeneous sources. In *Proc. of VLDB*, pages 256–265, Athens, Greece, 1997.

[Wie92] Gio Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, pages 38–49, 1992.

[YFIV00] Khaled Yagoub, Daniela Florescu, Valerie Issarny, and Patrick Valduriez. Caching strategies for data-intensive web sites. In *Proc. of VLDB*, pages 188–199, Cairo, Egypt, 2000.

[YKL97] J. Yang, K. Karlapalem, and Q. Li. Algorithms for materialized view design in data warehousing environment. In *Proc. of VLDB*, pages 136–145, Athens, Greece, 1997.

[YL87] H. Z. Yang and P. A. Larson. Query transformation for PSJ-queries. In *Proc. of VLDB*, pages 245–254, Brighton, England, 1987.

[ZCL$^+$00] Markos Zaharioudakis, Roberta Cochrane, George Lapis, Hamid Pirahesh, and Monica Urata. Answering complex SQL queries using automatic summary tables. In *Proc. of SIGMOD*, pages 105–116, 2000.

[ZO93] X. Zhang and M. Z. Ozsoyoglu. On efficient reasoning with implication constraints. In *Proc. of DOOD*, pages 236–252, 1993.

Data Exchange: Semantics and Query Answering

Ronald Fagin[1], Phokion G. Kolaitis[2]*, Renée J. Miller[3]*, and Lucian Popa[1]

[1] IBM Almaden Research Center
[2] UC Santa Cruz
[3] University of Toronto

Abstract. Data exchange is the problem of taking data structured under a source schema and creating an instance of a target schema that reflects the source data as accurately as possible. In this paper, we address foundational and algorithmic issues related to the semantics of data exchange and to query answering in the context of data exchange. These issues arise because, given a source instance, there may be many target instances that satisfy the constraints of the data exchange problem. We give an algebraic specification that selects, among all solutions to the data exchange problem, a special class of solutions that we call *universal*. A universal solution has no more and no less data than required for data exchange and it represents the entire space of possible solutions. We then identify fairly general, and practical, conditions that guarantee the existence of a universal solution and yield algorithms to compute a canonical universal solution efficiently. We adopt the notion of "certain answers" in indefinite databases for the semantics for query answering in data exchange. We investigate the computational complexity of computing the certain answers in this context and also study the problem of computing the certain answers of target queries by simply evaluating them on a canonical universal solution.

1 Introduction

In *data exchange*, data structured under one schema (which we call a *source schema*) must be restructured and translated into an instance of a different schema (a *target schema*). Data exchange is used in many tasks that require data to be transferred between existing, independently created applications. The first systems supporting the restructuring and translation of data were built several decades ago. An early such system was EXPRESS [21], which performed data exchange between hierarchical schemas. The need for systems supporting data exchange has persisted over the years. Recently this need has become more pronounced, as the terrain for data exchange has expanded with the proliferation of web data that are stored in different formats, such as traditional relational database schemas, semi-structured schemas (for example, DTDs or XML schemas), and various scientific formats. In this paper, we address several foundational and algorithmic issues related to the semantics of data exchange and to query answering in the context of data exchange.

* Research carried out while these authors were visiting scientists at the IBM Almaden Research Center. Kolaitis was also partially supported by NSF Grant IIS-9907419. Miller was also partially supported by a research grant from NSERC.

D. Calvanese et al. (Eds.): ICDT 2003, LNCS 2572, pp. 207–224, 2003.

The data exchange problem. In a data exchange setting, we have a source schema $\mathbf{S}$ and a target schema $\mathbf{T}$, where we assume that $\mathbf{S}$ and $\mathbf{T}$ are disjoint. Since $\mathbf{T}$ can be an independently created schema, it may have its own constraints that are given as a set Σ_t of sentences in some logical formalism over $\mathbf{T}$. In addition, we must have a way of modeling the relationship between the source and target schemas. This essential element of data exchange is captured by *source-to-target dependencies* that specify how and what source data should appear in the target. These dependencies are assertions between a source query and a target query. Formally, we have a set Σ_{st} of *source-to-target dependencies* of the form $\forall \mathbf{x}(\phi_{\mathbf{S}}(\mathbf{x}) \rightarrow \chi_{\mathbf{T}}(\mathbf{x}))$, where $\phi_{\mathbf{S}}(\mathbf{x})$ is a formula in some logical formalism over $\mathbf{S}$ and $\chi_{\mathbf{T}}(\mathbf{x})$ is a formula in some (perhaps different) logical formalism over $\mathbf{T}$.

Consider a data exchange setting determined by $\mathbf{S}$, $\mathbf{T}$, Σ_{st}, and Σ_t as above. This setting gives rise to the following *data exchange problem:* given an instance I over the source schema $\mathbf{S}$, materialize an instance J over the target schema $\mathbf{T}$ such that the target dependencies Σ_t are satisfied by J, and the source-to-target dependencies Σ_{st} are satisfied by I and J together. The first crucial observation is that there may be many solutions (or none) for a given instance of the data exchange problem. Hence, several conceptual and technical questions arise concerning the semantics of data exchange. First, when does a solution exist? If many solutions exist, which solution should we materialize and what properties should it have, so that it reflects the source data as accurately as possible? Finally, can such a "good" solution be efficiently computed?

We consider the semantics of the data exchange problem to be one of the two main issues in data exchange. We believe that the other main issue is query answering. Specifically, suppose that q is a query over the target schema $\mathbf{T}$ and I is an instance over the source schema $\mathbf{S}$. What does answering q with respect to I mean? Clearly, there is an ambiguity arising from the fact that, as mentioned earlier, there may be many solutions J for I and, as a result, different such solutions J may produce different answers $q(J)$. This conceptual difficulty was first encountered in the context of *incomplete* or *indefinite* databases (see, for instance, [23]), where one has to find the "right" answers to a query posed against a set of "possible" databases. There is general agreement that in the context of incomplete databases, the "right" answers are the *certain* answers, that is, the answers that occur in the intersection of all $q(J)$'s, as J varies over all "possible" databases. This notion makes good sense for data exchange as well, where the "possible" databases are the solutions J for the instance I. We thus adopt the certain answers for the semantics of query answering in the data exchange setting and investigate the complexity of computing the certain answers in the data exchange setting. A related important question is whether the certain answers of a query can be computed by query evaluation on the "good" target instance that we chose to materialize.

Data exchange vs. data integration. Before describing our results on data exchange, we briefly compare and contrast data exchange with *data integration*. Following the terminology and notation in the recent overview [13], a *data integration system* is a triple $\langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, where $\mathcal{G}$ is the *global schema*, $\mathcal{S}$ is the *source schema*, and $\mathcal{M}$ is a set of *assertions* relating elements of the global schema with elements of the source schema. Both $\mathcal{G}$ and $\mathcal{S}$ are specified in suitable languages that may allow for the expression of various constraints. In this generality, a data exchange setting $\langle \mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t \rangle$ can be thought of as a data integration system in which $\mathbf{S}$ is the source schema, $\mathbf{T}$ and Σ_t form the global schema, and the source-to-target dependencies in Σ_{st} are the assertions of the data integration system. In practice, however, most data integration systems studied to date are either LAV (*local-as-view*) systems or GAV (*global-as-view*) systems [11,13, 14]. In a LAV system, each assertion in $\mathcal{M}$ relates one element of the source schema $\mathcal{S}$ to a query (a view) over the global schema $\mathcal{G}$; moreover, it is usually assumed that there are no target constraints ($\Sigma_t = \emptyset$). In a GAV system the reverse holds, that is, each assertion in $\mathcal{M}$ relates one element of the global schema $\mathcal{G}$ to a query (a view) over the source schema $\mathcal{S}$. Since the source-to-target dependencies Σ_{st} relate a query over the source schema $\mathbf{S}$ to a query over the target schema $\mathbf{T}$, a data exchange setting is neither a LAV nor a GAV system. Instead, it can be thought of as a GLAV (*global-and-local-as-view*) system [10,13].

The above similarities notwithstanding, there are important differences between data exchange and data integration. As mentioned earlier, in data exchange scenarios, the target schema is often independently created and comes with its own constraints. In data integration, however, the global schema $\mathcal{G}$ is commonly assumed to be a reconciled, virtual view of a heterogeneous collection of sources and, as such, has no constraints. In fact, with the notable exception of [5], which studied the impact of key and foreign key constraints on query answering in a GAV system, most research on data integration has not taken target constraints into account. Still, a more significant difference is that in a data exchange setting we have to actually materialize a finite target instance that best reflects the given source instance. In data integration no such exchange of data is required. For query answering, both data exchange and data integration use the certain answers as the standard semantics of queries over the target (global) schema. In data integration, the source instances are used to compute the certain answers of queries over the global schema. In contrast, in a data exchange setting, it may not be feasible to couple applications together in a manner that data may be retrieved and shared on-demand at query time. This may occur, for instance, in peer-to-peer applications that must share data, yet maintain autonomy. Hence, queries over the target schema may have to be answered using the materialized target instance alone, without reference to the original source instance. This leads to the following problem in data exchange: under what conditions and for which queries can the certain answers be computed using just the materialized target instance?

Motivation from Clio. The results presented here were motivated by our experience with Clio, a prototype schema mapping and data exchange tool to whose development some of us have contributed [18,19]. In Clio, source-to-target dependencies are (semi)-automatically generated from a set of correspondences between the source schema and the target schema; these dependencies can then be used in a data integration system to compute the certain answers to target queries. Most of the applications we considered, however, were decoupled applications that would have had to be rewritten to operate cooperatively, as required in data integration. For this reason, early on in the development of Clio, we recognized the need to go farther and, given a source instance, generate a single "universal" target instance that was the result of the schema mapping. In designing the algorithms used in Clio for creating the target instance, we were guided mainly by our own intuition rather than by formal considerations. It should be noted that there is a long history of work on data translation that focuses on taking high-level, data-independent translation rules and generating efficient, executable translation programs [1,20,21]. Yet, we could not find a formal justification for the intuitive choices we made in creating

the target instance. In seeking to formalize this intuition and justify the choices made in Clio, we were led to explore foundational and algorithmic issues related to the semantics of data exchange and query answering in this setting. Clio supports schemas that are relational or semi-structured. However, challenging issues already arise in the relational case. For this reason, here we focus exclusively on data exchange between relational schemas; extending this work to other types of schemas is the subject of on-going investigation.

Summary of Results. In Section 2, we formally introduce the data exchange problem. We then give an algebraic specification that selects, among all possible solutions for a given source instance, a special class of solutions that we call *universal*. More precisely, a solution for an instance of the data exchange problem is universal if it has homomorphisms to all solutions for that instance. We show that a universal solution has "good" properties that justify its choice for the semantics of the data exchange problem. We note that Calì et al. [5] studied GAV systems with key and foreign key constraints at the target. By means of a logic program that simulates the foreign key constraints, they constructed a *canonical database*, which turns out to be a particular example of our notion of universal solution.

Given the declarative specification of universal solutions, we go on in Section 3 to identify fairly general, yet practical, sufficient conditions that guarantee the existence of a universal solution and yield algorithms to compute such a solution efficiently. We introduce the concept of a *weakly acyclic* set of target dependencies, which is broad enough to contain as special cases both sets of full tuple-generating dependencies (full tgds) [4] and acyclic sets of inclusion dependencies [7]. We then show that if Σ_{st} is a set of tuple-generating dependencies (tgds) and Σ_t is the union of a weakly acyclic set of tgds with a set of equality generating dependencies (egds), then, given a source instance of the data exchange problem, (1) a universal solution exists if and only if a solution exists, and (2) there is a polynomial-time algorithm that determines whether a solution exists and, if so, it produces a particular universal solution, which we call the *canonical* universal solution. These results make use of the classical *chase* procedure [4,15]. We note that, even though the chase has been widely used in reasoning about dependencies, we have not been able to find any explicit references to the fact that the chase can produce instances that have homomorphisms to all instances satisfying the dependencies under consideration.

After this, in Sections 4 and 5, we address algorithmic issues related to query answering in the data exchange setting. We study the computational complexity of computing the certain answers, and explore the boundary of what queries can and cannot be answered in a data exchange setting using the exchanged target instance alone. On the positive side, if q is a union of conjunctive queries, then it is easy to show that the certain answers of q can indeed be obtained by evaluating q on an arbitrary universal solution. Moreover, universal solutions are the only solutions possessing this property; this can be seen as further justification for our choice to use universal solutions for data exchange. It also follows that, whenever a universal solution can be computed in polynomial time, the certain answers of unions of conjunctive queries can be computed in polynomial time (in particular, this is true when the dependencies in Σ_{st} and Σ_t satisfy the conditions identified in Section 3).

On the negative side, a dramatic change occurs when queries have inequalities. The hardness of this problem arises from the complexity of reasoning over uncertain databases, not from the data exchange *per se*. Indeed, Abiteboul and Duschka [2] showed that in a LAV data integration system and with conjunctive queries as views, computing the certain answers of conjunctive queries with inequalities is a coNP-complete problem. Since LAV is a special case of a data exchange setting in which the canonical universal solution can be computed in polynomial time, it follows that, unless $\mathrm{P} = \mathrm{NP}$, we cannot compute the certain answers of conjunctive queries with inequalities by evaluating them on the canonical universal solution (or on any other polynomial-time computable universal solution).

We take a closer look at conjunctive queries with inequalities by focusing on the number of inequalities. In [2], it was claimed that in the LAV setting and with conjunctive queries as views, computing the certain answers of conjunctive queries with a single inequality is a coNP-hard problem. The reduction given in that paper, however, is not correct; a different reduction in the unpublished full version [3] shows that computing the certain answers of conjunctive queries with six (or more) inequalities is a coNP-complete problem. We conjecture that the minimum number of inequalities that give rise to such coNP-hardness results is two. Towards this, we show that in the same LAV setting, computing the certain answers of *unions* of conjunctive queries with at most two inequalities per disjunct is coNP-complete. This result is tight, because we show that, even for the more general data exchange setting, there is a polynomial-time algorithm for computing the certain answers of unions of conjunctive queries with at most one inequality per disjunct (thus, the claim in [2] is false, unless $\mathrm{P} = \mathrm{NP}$). Moreover, the certain answers of unions of conjunctive queries with at most one inequality per disjunct can be computed in time polynomial in the size of a given universal solution. We point out, however, that this computation cannot be carried out by simply evaluating such queries on the canonical universal solution. Thus, the question arises as to whether the certain answers of unions of conjunctive queries with at most one inequality per disjunct can be computed by evaluating some other (perhaps more complex) first-order query on the canonical universal solution. Our final theorem provides a strong negative answer to this question. It shows that there is a simple conjunctive query q with one inequality for which there is no first-order query q^* such that the certain answers of q can be computed by evaluating q^* on the canonical universal solution. The proof of this theorem makes use of a novel combination of Ehrenfeucht-Fraïssé games and the chase.

The proofs of the results of this paper can be found in the full version [9].

2 The Data Exchange Problem

A *schema* is a finite collection $\mathbf{R} = \{R_1, \ldots, R_k\}$ of relation symbols. An *instance I over the schema* $\mathbf{R}$ is a function that associates to each relation symbol R_i a relation $I(R_i)$. set of relations $I(R_1), \ldots, I(R_k)$ interpreting the corresponding relation symbols in $\mathbf{S}$. In the sequel, we will on occasion abuse the notation and use R_i to denote both the relation symbol and the relation that interprets it. Given a tuple t occurring in a relation R, we denote by $R(t)$ the association between t and R and call it a *fact*. If $\mathbf{R}$ is a schema, then a *dependency over* $\mathbf{R}$ is a sentence in some logical formalism over $\mathbf{R}$.

Let $\mathbf{S} = \{S_1, \ldots, S_n\}$ and $\mathbf{T} = \{T_1, \ldots, T_m\}$ be two disjoint schemas. We refer to $\mathbf{S}$ as the *source* schema and to the S_i's as the *source* relation symbols. We refer to $\mathbf{T}$ as the *target* schema and to the T_j's as the *target* relation symbols. Similarly, instances over $\mathbf{S}$ will be called *source* instances, while instances over $\mathbf{T}$ will be called *target* instances. If I is a source instance and J is a target instance, then we write $\langle I, J\rangle$ for the instance K over the schema $\mathbf{S} \cup \mathbf{T}$ such that $K(S_i) = I(S_i)$ and $K(T_j) = J(T_j)$, for $i \leq n$ and $j \leq m$.

A *source-to-target dependency* is a dependency of the form $\forall \mathbf{x}(\phi_{\mathbf{S}}(\mathbf{x}) \rightarrow \chi_{\mathbf{T}}(\mathbf{x}))$, where $\phi_{\mathbf{S}}(\mathbf{x})$ is a formula, with free variables $\mathbf{x}$, of some logical formalism over $\mathbf{S}$ and $\chi_{\mathbf{T}}(\mathbf{x})$ is a formula, with free variables $\mathbf{x}$, over some logical formalism over $\mathbf{T}$ (these two logical formalisms may be different). We use the notation $\mathbf{x}$ for a vector of variables $x_1, \ldots, x_k$. A *target* dependency is a dependency over the target schema $\mathbf{T}$ (the formalism used to express a target dependency may be different from those used for the source-to-target dependencies). The source schema may also have dependencies that we assume are satisfied by every source instance. While the source dependencies may play an important role in deriving source-to-target dependencies [19], they do not play any direct role in data exchange, because we take the source instance to be given.

Definition 1. A *data exchange setting* $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ consists of a source schema $\mathbf{S}$, a target schema $\mathbf{T}$, a set Σ_{st} of source-to-target dependencies, and a set Σ_t of target dependencies. The *data exchange problem* associated with this setting is the following: given a finite source instance I, find a finite target instance J such that $\langle I, J\rangle$ satisfies Σ_{st} and J satisfies Σ_t. Such a J is called a *solution for I* or, simply a *solution* if the source instance I is understood from the context. The set of all solutions for I is denoted by $\mathrm{Sol}(I)$.

For most practical purposes, and for most of the results of this paper, each source-to-target dependency in Σ_{st} is a *tuple generating dependency (tgd)* [4] of the form

$$\forall \mathbf{x}(\phi_{\mathbf{S}}(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y})),$$

where $\phi_{\mathbf{S}}(\mathbf{x})$ is a conjunction of atomic formulas over $\mathbf{S}$ and $\psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y})$ is a conjunction of atomic formulas over $\mathbf{T}$. Moreover, each target dependency in Σ_t is either a *tuple-generating dependency (tgd)* (of the form shown below left) or an *equality-generating dependency (egd)* [4] (shown below right):

$$\forall \mathbf{x}(\phi_{\mathbf{T}}(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y})) \qquad \forall \mathbf{x}(\phi_{\mathbf{T}}(\mathbf{x}) \rightarrow (x_1 = x_2))$$

In the above, $\phi_{\mathbf{T}}(\mathbf{x})$ and $\psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y})$ are conjunctions of atomic formulas over $\mathbf{T}$, and x_1, x_2 are among the variables in $\mathbf{x}$. Note that data exchange settings with tgds as source-to-target dependencies include as special cases both LAV and GAV data integration systems in which the views are sound [13] and are defined by conjunctive queries. It is natural to take the target dependencies to be tgds and egds: these two classes together comprise the (embedded) implicational dependencies [8], which seem to include essentially all of the naturally-occurring constraints on relational databases. However, it is somewhat surprising that tgds, which were originally "designed" for other purposes (as constraints), turn out to be ideally suited for describing desired data transfer. For simplicity, in the rest of the paper we will drop the universal quantifiers in front of a dependency, and implicitly assume such quantification. However, we will write down all existential quantifiers.

The next example shows that there may be more than one possible solution for a given data exchange problem. The natural question is then which solution to choose.

Example 1. Consider a data exchange problem in which the source schema has three relation symbols P, Q, R, each of them with attributes A, B, C, while the target schema has one relation symbol T also with attributes A, B, C. We assume that $\Sigma_t = \emptyset$. The source-to-target dependencies and the source instance are:

$$\begin{aligned} \Sigma_{st}: \ & P(a,b,c) \rightarrow \exists Y \exists Z\, T(a,Y,Z) \\ & Q(a,b,c) \rightarrow \exists X \exists U\, T(X,b,U) \\ & R(a,b,c) \rightarrow \exists V \exists W\, T(V,W,c) \end{aligned} \qquad \begin{aligned} I = \{ \ & P(a_0, b_0', c_0'), \\ & Q(a_0'', b_0, c_0''), \\ & R(a_0''', b_0''', c_0) \ \} \end{aligned}$$

We observe first that the dependencies in Σ_{st} do not completely specify the target instance. It should be noted that such incomplete specification arises naturally in many practical scenarios of data exchange (or data integration for that matter; see [11,13]). For our example, one possible solution is:

$$J = \{T(a_0, Y_0, Z_0), T(X_0, b_0, U_0), T(V_0, W_0, c_0)\},$$

where $X_0, Y_0, \ldots$ represent "unknown" values. We will call such values *labeled nulls* and we will introduce them formally in the next section. The second observation is that there may be more than one solution. For example, the following are solutions as well:

$$J_1 = \{T(a_0, b_0, c_0)\} \qquad J_2 = \{T(a_0, b_0, Z_1), T(V_1, W_1, c_0)\}$$

In the above, Z_1, V_1 and W_1 are labeled nulls. Note that J_1 does not use labeled nulls; instead, source values are used to witness the existentially quantified variables in the dependencies. Solution J_1 seems to be less general than J, since it "assumes" that all three tuples required by the dependencies are equal to the tuple (a_0, b_0, c_0). This assumption, however, is not part of the specification. Similarly, solution J_2 has extra information that is not a consequence of the dependencies in Σ_{st} for the given source data. We argue that neither J_1 nor J_2 should be used for data exchange. In contrast, J is the "best" solution: it contains no more and no less than what the specification requires. We formalize this intuition next.

2.1 Universal Solutions

We next give an algebraic specification that selects, among all possible solutions, a special class of solutions that we call *universal*. As we will see, a universal solution has several "good" properties that justify its choice for the semantics of data exchange. Before presenting the key definition, we introduce some terminology and notation.

We denote by $\underline{\mathsf{Const}}$ the set of all values that occur in source instances and we also call them *constants*. In addition, we assume an infinite set $\underline{\mathsf{Var}}$ of values, which we call *labeled nulls*, such that $\underline{\mathsf{Var}} \cap \underline{\mathsf{Const}} = \emptyset$. We reserve the symbols $I, I', I_1, I_2, \ldots$ for instances over the source schema I and with values in $\underline{\mathsf{Const}}$. We also reserve the symbols $J, J', J_1, J_2, \ldots$ for instances over the target schema $\mathbf{T}$ and with values in $\underline{\mathsf{Const}} \cup \underline{\mathsf{Var}}$.

If $\mathbf{R} = \{R_1, \ldots, R_k\}$ is a schema and K is an instance over $\mathbf{R}$ with values in $\underline{\mathsf{Const}} \cup \underline{\mathsf{Var}}$, then $\underline{\mathsf{Var}}(K)$ denotes the set of labelled nulls occurring in relations in K.

Definition 2. Let K_1 and K_2 be two instances over $\mathbf{R}$ with values in $\underline{\mathsf{Const}} \cup \underline{\mathsf{Var}}$.

A *homomorphism* $h : K_1 \rightarrow K_2$ is a mapping from $\underline{\mathsf{Const}} \cup \underline{\mathsf{Var}}(K_1)$ to $\underline{\mathsf{Const}} \cup \underline{\mathsf{Var}}(K_2)$ such that: (1) $h(c) = c$, for every $c \in \underline{\mathsf{Const}}$; (2) for every fact $R_i(t)$ of K_1, we have that $R_i(h(t))$ is a fact of K_2 (where, if $t = (a_1, \ldots, a_s)$, then $h(t) = (h(a_1), \ldots, h(a_s))$).

K_1 is *homomorphically equivalent* to K_2 if there is a homomorphism $h : K_1 \rightarrow K_2$ and a homomorphism $h' : K_2 \rightarrow K_1$.

Definition 3 (Universal solution). Consider a data exchange setting $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$. If I is a source instance, then a *universal solution for* I is a solution J for I such that for every solution J' for I, there exists a homomorphism $h : J \rightarrow J'$.

Example 2. The instances J_1 and J_2 in Example 1 are not universal. In particular, there is no homomorphism from J_1 to J and also there is no homomorphism from J_2 to J. This fact makes precise our earlier intuition that the instances J_1 and J_2 contain "extra" information. In contrast, there exist homomorphisms from J to both J_1 and J_2. Actually, it can be easily shown that J has homomorphisms to all other solutions. Thus, J is universal.

From an algebraic standpoint, being a universal solution is a property akin to being an *initial structure* [17] for the set of all solutions (although an initial structure for a set $\mathcal{K}$ of structures is required to have *unique* homomorphisms to all other structures in $\mathcal{K}$). Initial structures are ubiquitous in several areas of computer science, including semantics of programming languages and term rewriting, and are known to have good properties (see [17]). The next result asserts that universal solutions have good properties as well.

Proposition 1. *Let $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ be a data exchange setting.*

1. *If I is a source instance and J, J' are universal solutions for I, then J and J' are homomorphically equivalent.*
2. *Assume that Σ_{st} is a set of tgds. Let I, I' be two source instances, J a universal solution for I, and J' a universal solution for I'. Then $\mathrm{Sol}(I) \subseteq \mathrm{Sol}(I')$ if and only if there is a homomorphism $h : J' \rightarrow J$. Consequently, $\mathrm{Sol}(I) = \mathrm{Sol}(I')$ if and only if J and J' are homomorphically equivalent.*

The first part of Proposition 1 asserts that universal solutions are unique up to homomorphic equivalence. The second part implies that if J is a universal solution for two source instances I and I', then $\mathrm{Sol}(I) = \mathrm{Sol}(I')$. Thus, in a certain sense, each universal solution precisely embodies the space of solutions.

3 Computing Universal Solutions

Checking the conditions in Definition 3 requires implicitly the ability to check the (infinite) space of all solutions. Thus, it is not clear, at first hand, to what extent the notion of universal solution is a computable one. This section addresses the question of how to check the existence of a universal solution and how to compute one (if one exists). In particular, we show that the classical chase can be used for data exchange and that every finite chase, if it does not fail, constructs a universal solution. If the chase fails, then no solution exists. However, in general, for arbitrary dependencies, there may not exist a finite chase. Hence, in Section 3.2 we introduce the class of weakly acyclic sets of tgds, for which the chase is guaranteed to terminate in polynomial time. For such dependencies, we show that: (1) the existence of a universal solution can be checked in polynomial time, (2) a universal solution exists if and only if a solution exists, and (3) a universal solution (if solutions exist) can be produced in polynomial time.

3.1 Chase: Canonical Generation of Universal Solutions

Intuitively, we apply the following procedure to produce a universal solution: start with an instance $\langle I, \emptyset\rangle$ that consists of I, for the source schema, and of the empty instance, for the target schema; then chase $\langle I, \emptyset\rangle$ by applying the dependencies in Σ_{st} and Σ_t for as long as they are applicable[1]. This process may fail (as we shall see shortly, if an attempt to identify two constants is made) or it may never terminate. But if it does terminate and if it does not fail, then the resulting instance is guaranteed to satisfy the dependencies and, moreover, to be universal (Theorem 1).

We next define chase steps. Similar to homomorphisms between instances, a homomorphism from a conjunctive formula $\phi(\mathbf{x})$ to an instance J is a mapping from the variables $\mathbf{x}$ to $\underline{\mathsf{Const}} \cup \underline{\mathsf{Var}}(J)$ such that for every atom $R(x_1, \ldots, x_n)$ of ϕ, the fact $R(h(x_1), \ldots, h(x_n))$ is in J. The chase that we use is a slight variation of the classical notion of chase with tgds and egds of [4] in the sense that we chase instances rather than symbolic tableaux. Consequently, the chase may fail.

Definition 4 (Chase step). Let K be an instance.

(tgd) Let d be a tgd $\phi(\mathbf{x}) \to \exists \mathbf{y}\psi(\mathbf{x}, \mathbf{y})$. Let h be a homomorphism from $\phi(\mathbf{x})$ to K such that there is no extension of h to a homomorphism h' from $\phi(\mathbf{x}) \wedge \psi(\mathbf{x}, \mathbf{y})$ to K. We say that *d can be applied to K with homomorphism h*.

Let K' be the union of K with the set of facts obtained by: (a) extending h to h' such that each variable in $\mathbf{y}$ is assigned a fresh labeled null, followed by (b) taking the image of the atoms of ψ under h'. We say that *the result of applying d to K with h* is K', and write $K \xrightarrow{d,h} K'$.

(egd) Let d be an egd $\phi(\mathbf{x}) \to (x_1 = x_2)$. Let h be a homomorphism from $\phi(\mathbf{x})$ to K such that $h(x_1) \neq h(x_2)$. We say that *d can be applied to K with homomorphism h*. We distinguish two cases.

- If both $h(x_1)$ and $h(x_2)$ are in $\underline{\mathsf{Const}}$ then we say that *the result of applying d to K with h* is "failure", and write $K \xrightarrow{d,h} \perp$.
- Otherwise, let K' be K where we identify $h(x_1)$ and $h(x_2)$ as follows: if one is a constant, then the labeled null is replaced everywhere by the constant; if both are labeled nulls, then one is replaced everywhere by the other. We say that *the result of applying d to K with h* is K', and write $K \xrightarrow{d,h} K'$.

[1] It is possible to apply first Σ_{st} as long as applicable and then apply Σ_t as long as applicable.

In the definition, $K \xrightarrow{d,h} K'$ (including the case where K' is $\perp$) defines one single chase step. We next define chase sequences and finite chases.

Definition 5 (Chase). Let Σ be a set of tgds and egds, and let K be an instance.

- A *chase sequence of K with Σ* is a sequence (finite or infinite) of chase steps $K_i \xrightarrow{d_i,h_i} K_{i+1}$, with $i = 0, 1, \ldots$, with $K = K_0$ and d_i a dependency in Σ.
- A *finite chase of K with Σ* is a finite chase sequence $K_i \xrightarrow{d,h} K_{i+1}, 0 \leq i < m$, with the requirement that either (a) $K_m = \perp$ or (b) there is no dependency d_i of Σ and there is no homomorphism h_i such that d_i can be applied to K_m with h_i. We say that K_m is the result of the finite chase. We refer to case (a) as the case of a *failing finite chase* and we refer to case (b) as the case of a *successful finite chase*.

In general, there may not exist a finite chase of an instance (cyclic sets of dependencies could cause infinite application of chase steps). Infinite chases can be defined as well, but for this paper we do not need to do so. Also, different chase sequences may yield different results. However, each result, if not $\perp$, satisfies Σ.

For data exchange, we note first that, due to the nature of our dependencies, any chase sequence that starts with $\langle I, \emptyset\rangle$ does not change or add tuples in I. Then, if a finite chase exists, its result $\langle I, J\rangle$ is such that J is a solution. Furthermore, J is universal, a fact that does not seem to have been explicitly noted in the literature on the chase. The next theorem states this, and also states that the chase can be used to check the existence of a solution.

Theorem 1. *Assume a data exchange setting where Σ_{st} consists of tgds and Σ_t consists of tgds and egds.*

1. *Let $\langle I, J\rangle$ be the result of some successful finite chase of $\langle I, \emptyset\rangle$ with $\Sigma_{st} \cup \Sigma_t$. Then J is a universal solution.*
2. *If there exists some failing finite chase of $\langle I, \emptyset\rangle$ with $\Sigma_{st} \cup \Sigma_t$, then there is no solution.*

For case 1 of Theorem 1 we refer to such J as *a canonical universal solution*. In further examples and proofs, when such J is unique, we will also use the term *the canonical universal solution*. We note that a canonical universal solution is similar, in its construction, to the representative instance defined in the work on the universal relation (see [16]).

The following is an example of cyclic set of inclusion dependencies for which there is no finite chase; thus, we cannot produce a universal solution by the chase. Still, a finite solution does exist. This illustrates the need for introducing restrictions in the class of dependencies that are allowed in the target.

Example 3. Consider the data exchange setting $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ as follows. The source schema $\mathbf{S}$ has one relation `DeptEmp(dpt_id, mgr_name, eid)` listing departments with their managers and their employees. The target schema $\mathbf{T}$ has a relation `Dept(dpt_id, mgr_id, mgr_name)` for departments and their managers, and a separate relation for employees `Emp(eid, dpt_id)`. The source-to-target and target dependencies are:

$$\Sigma_{st} = \{\ \mathtt{DeptEmp}(d, n, e) \to \exists M.\mathtt{Dept}(d, M, n) \wedge \mathtt{Emp}(e, d)\ \}$$
$$\Sigma_t = \{\ \mathtt{Dept}(d, m, n) \to \exists D.\mathtt{Emp}(m, D), \quad \mathtt{Emp}(e, d) \to \exists M \exists N.\mathtt{Dept}(d, M, N)\ \}$$

Assume now that the source instance I has one tuple in DeptEmp, for department *CS* with manager *Mary* and employee *E003*. Chasing $\langle I, \emptyset \rangle$ with Σ_{st} yields the target instance:

$$J_1 = \{\texttt{Dept}(CS, M, Mary), \texttt{Emp}(E003, CS)\}$$

where M is a labeled null that instantiates the existentially quantified variable of the tgd, and encodes the unknown manager id of *Mary*. However, J_1 does not satisfy Σ_t; therefore, the chase does not stop at J_1. The first tgd in Σ_t requires M to appear in Emp as an employee id. Thus, the chase will add $\texttt{Emp}(M, D)$ where D is a labeled null representing the unknown department in which Mary is employeed. Then the second tgd becomes applicable, and so on. It is easy to see that there is no finite chase. Satisfying all the dependencies would require building an infinite instance:

$$J = \{\texttt{Dept}(CS, M, Mary), \texttt{Emp}(E003, CS), \texttt{Emp}(M, D), \texttt{Dept}(D, M', N'), \dots\}$$

On the other hand, finite solutions exist. Two such examples are:

$$J' = \{\texttt{Dept}(CS, E003, Mary), \texttt{Emp}(E003, CS)\}$$
$$J'' = \{\texttt{Dept}(CS, M, Mary), \texttt{Emp}(E003, CS), \texttt{Emp}(M, CS)\}$$

However, neither J' nor J'' are universal: there is no homomorphism from J' to J'' and there is no homomorphism from J'' to J'. We argue that neither should be used for data exchange. In particular, J' makes the assumption that the manager id of *Mary* is equal to *E003*, while J'' makes the assumption that the department in which *Mary* is employed is the same as the department (*CS*) that *Mary* manages. Neither assumption is a consequence of the given dependencies and source instance. It can be shown that no *finite* universal solution exists for this example.

We next consider sets of dependencies for which every chase sequence is guaranteed to reach its end after at most polynomially many steps (in the size of the input instance). For such sets of dependencies it follows that checking the existence of a solution, as well as generating a universal solution, can be carried out in polynomial time.

3.2 Polynomial-Length Chase

We first discuss sets of *full tgds* (tgds with no existentially quantified variables). It has been proven in [4] that every chase sequence with a set Σ of full tgds has at most finite length. Moreover every chase has the same result. It is simple to show that the length of the chase is bounded by a polynomial in the size of the input instance (the dependencies and the schema are fixed). Also, any set of egds can be added to Σ without affecting the uniqueness of the result or the polynomial bound.

Although full tgds enjoy nice properties, they are not very useful in practice. Most dependencies occurring in real schemas are non-full, for example, foreign key constraints or, more generally, inclusion dependencies [6]. It is well known that chasing with inclusion dependencies may not terminate in general. *Acyclic sets of inclusion dependencies* [7] are a special case for which every chase sequence has a length that is polynomial in the size of the input instance. Such dependencies can be described by defining a directed graph in which the nodes are the relation symbols, and such that

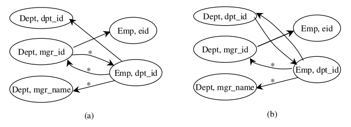

Fig. 1. Dependency graphs for: (a) a set of tgds that is not weakly acyclic, (b) a weakly acyclic set of tgds.

there exists an edge from R to S whenever there is an inclusion dependency from R to S. A set of inclusion dependencies is acyclic if there is no cycle in this graph. We define next a *weakly acyclic sets of tgds*, a notion that strictly includes both sets of full tgds and acyclic sets of inclusion dependencies. This notion is inspired by the definition of weakly recursive ILOG [12], even though the latter is not directly related to dependencies. Informally, a set of tgds is weakly acyclic if it does not allow for cascading of labeled null creation during the chase.

Definition 6 (Weakly acyclic set of tgds). Let Σ be a set of tgds over a fixed schema. Construct a directed graph, called the *dependency graph*, as follows: (1) there is a node for every pair (R, A) with R a relation symbol of the schema and A an attribute of R; call such pair (R, A) a *position*; (2) add edges as follows: for every tgd $\phi(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi(\mathbf{x}, \mathbf{y})$ in Σ and for every x in $\mathbf{x}$ that **occurs** in ψ:

- For every occurence of x in ϕ in position (R, A_i):
  (a) for every occurence of x in ψ in position (S, B_j), add an edge $(R, A_i) \rightarrow (S, B_j)$ (if it does not already exist).
  (b) in addition, for every existentially quantified variable y and for every occurrence of y in ψ in position (T, C_k), add a *special edge* $(R, A_i) \xrightarrow{*} (T, C_k)$ (if it does not already exist).

Note that there may be two edges in the same direction between two nodes, if exactly one of the two edges is special. Then Σ is *weakly acyclic* if the dependency graph has no cycle going through a special edge. ∎

Intuitively, part (a) keeps track of the fact that a value may propagate from position (R, A_i) to position (S, B_j) during the chase. Part (b), moreover, keeps track of the fact that propagation of a value into (S, B_j) also creates a labeled null in any position that has an existentially quantified variable. If a cycle goes through a special edge, then a labeled null appearing in a certain position during the chase may determine the creation of another labeled null, in the same position, at a later chase step. This process may thus continue forever. Note that the definition allows for cycles as long as they do not include special edges. In particular, a set of full tgds is a special case of a weakly acyclic set of tgds (there are no existentially quantified variables, and hence no special edges).

Example 4. Recall Example 3. The dependency graph of Σ_t is shown in Figure 1(a). The graph contains a cycle with two special edges. Hence Σ_t is not weakly acyclic and therefore a finite chase may not exist (as seen in Example 3). On the other hand, let us assume that we know that each manager of a department is employed by the *same* department. Then, we replace the set Σ_t by the set Σ_t', where

$$\Sigma'_t = \{ \texttt{Dept}(d,m,n) \rightarrow \texttt{Emp}(m,d),\ \texttt{Emp}(e,d) \rightarrow \exists M \exists N.\texttt{Dept}(d,M,N) \}$$

The dependency graph of Σ'_t, shown in Figure 1(b), has no cycles going through a special edge. Thus, Σ'_t is weakly acyclic. As Theorem 2 will show, it is guaranteed that every chase sequence is finite. For Example 3, one can see that the chase of J_1 with Σ'_t stops with result J''. Thus J'' is universal. Note that for J'' to be universal it was essential that we explicitly encoded in the dependencies the fact that managers are employed by the department they manage. Finally, we remark that Σ'_t is an example of a set of inclusion dependencies that, although weakly acyclic, is cyclic according to the definition of [7].

We now state the main results regarding weakly acyclic sets of tgds.

Theorem 2. *Let Σ be the union of a weakly acyclic set of tgds with a set of egds, and let K be an instance. Then there exists a polynomial in the size of K that bounds the length of every chase sequence of K with Σ.*

Corollary 1. *Assume a data exchange setting where Σ_{st} is a set of tgds, and Σ_t is the union of a weakly acyclic set of tgds with a set of egds. The existence of a solution can be checked in polynomial time. If a solution exists, then a universal solution can be produced in polynomial time.*

4 Query Answering

As stated earlier, we adopt the notion of certain answers for the semantics of query answering. We first give the formal definition of this notion and then address the problem of whether and to what extent the certain answers of a query over the target schema can be computed by evaluating some query (same or different) on a universal solution.

Definition 7. Let $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ be a data exchange setting.

- Let q be a k-ary query over the target schema $\mathbf{T}$ and I a source instance. The *certain answers of q with respect to I*, denoted by *certain*(q, I), is the set of all k-tuples t of constants from I such that for every solution J of this instance of the data exchange problem, we have that $t \in q(J)$.
- Let q be a Boolean query over the target schema $\mathbf{T}$ and I a source instance. We write *certain*$(q, I) = \underline{\mathsf{true}}$ to denote that for every solution J of this instance of the data exchange problem, we have that $q(J) = \underline{\mathsf{true}}$. We also write *certain*$(q, I) = \underline{\mathsf{false}}$ to denote that there is a solution J such that $q(J) = \underline{\mathsf{false}}$.

On the face of it, the definition of certain answers entails a computation over the entire set of solutions of a given instance of the data exchange problem. Since this set may very well be infinite, it is desirable to identify situations in which the certain answers of a query q can be computed by evaluating q on a particular fixed solution and then keeping only the tuples that consist entirely of constants. More formally, if q is a k-ary query and J is a target instance, then $q(J)_\downarrow$ is the set of all k-tuples t of constants such that $t \in q(J)$. We extend the notation to Boolean queries by agreeing that if q is a Boolean query, then $q(J)_\downarrow = q(J)$ $(= \underline{\mathsf{true}}$ or $\underline{\mathsf{false}})$.

The next proposition characterizes universal solutions with respect to query answering, when the queries under consideration are unions of conjunctive queries. First, it

shows that *certain*$(q, I) = q(J)_\downarrow$ whenever J is a universal solution and q is a union of conjunctive queries. Concrete instances of this result in the LAV setting have been established in [2]. Another instance of this result has also been noted for the GAV setting with key/foreign key constraints in [5]. Thus, evaluation of conjunctive queries on an arbitrarily chosen universal solution gives precisely the set of certain answers. Moreover, the second statement of the proposition shows that the universal solutions are the only solutions that have this property. This is further justification for using universal solutions for data exchange.

Proposition 2. *Consider a data exchange setting with* $\mathbf{S}$ *as the source schema,* $\mathbf{T}$ *as the target schema, and such that the dependencies in the sets Σ_{st} and Σ_t are arbitrary.*

1. *Let q be a union of conjunctive queries over the target schema* $\mathbf{T}$. *If I is a source instance and J is a universal solution, then certain$(q, I) = q(J)_\downarrow$.*
2. *Let I be a source instance and J be a solution such that for every conjunctive query q over* $\mathbf{T}$*, we have that certain$(q, I) = q(J)_\downarrow$. Then J is a universal solution.*

The following result follows from Corollary 1 and Part 1 of Proposition 2.

Corollary 2. *Assume a data exchange setting where Σ_{st} is a set of tgds, and Σ_t is the union of a weakly acyclic set of tgds with a set of egds. Let q be a union of conjunctive queries. For every source instance I, the set certain(q, I) can be computed in polynomial time in the size of I.*

The state of affairs changes dramatically when conjunctive queries with inequalities are considered. The next proposition shows that there is a simple Boolean conjunctive query q with inequalities such that no universal solution can be used to obtain the certain answers of q by evaluating q on that universal solution. This proposition also shows that in this particular case, there is another conjunctive query q^* with inequalities such that the certain answers of q can be obtained by evaluating q^* on the canonical universal solution.

Proposition 3. *Let S be a binary source relation symbol, T a binary target relation symbol, $S(x, y) \rightarrow \exists z(T(x, z) \wedge T(z, y))$ a source-to-target dependency, and q the following Boolean conjunctive query with one inequality: $\exists x \exists y(T(x, y) \wedge (x \neq y))$.*

1. *There is a source instance I such that certain$(q, I) = \underline{\mathsf{false}}$, but $q(J) = \underline{\mathsf{true}}$ for every universal solution J.*
2. *Let q^* be the query $\exists x \exists y \exists z(T(x, z) \wedge T(z, y) \wedge (x \neq y))$. If I is a source instance and J is the canonical universal solution, then certain$(q, I) = q^*(J)$.*

In view of Proposition 3, we address next the question of whether, given a conjunctive query with inequalities, it is always possible to find a query (not necessarily the same) that computes the certain answers when evaluated on the canonical universal solution.

5 Query Answering: Complexity and Inexpressibility

It is known that in LAV data integration systems, computing the certain answers of conjunctive queries with inequalities is a coNP-hard problem [2]. It follows that in the data exchange setting, it is not possible to compute the certain answers of such queries by evaluating them on the canonical universal solution or on any universal solution that is generated in polynomial time (unless $\mathrm{P} = \mathrm{NP}$). We take in Section 5.1 a closer look at conjunctive queries with inequalities. First we show (Theorem 3) that, in the data exchange setting, the problem of computing the certain answers for unions of conjunctive queries with inequalities is in coNP. Surprisingly, we show (Theorem 6) that there is a polynomial-time algorithm that computes the certain answers of unions of conjunctive queries with at most one inequality per disjunct. This is an optimal result because we also show (Theorem 5) that it is coNP-hard to compute the certain answers of unions of conjunctive queries with at most two inequalities per disjunct.

In the case of unions of conjunctive queries with at most one inequality per disjunct, the certain answers can be computed in polynomial time from an arbitrary universal solution. However, Section 5.2 shows (with no unproven complexity-theoretic assumptions such as $\mathrm{P} \neq \mathrm{NP}$) that there is a conjunctive query q with one inequality whose certain answers cannot be computed by rewriting q to a first-order query q^* and then evaluating q^* on the canonical universal solution. We begin by formally introducing the decision problem associated with the computation of the set of certain answers.

Definition 8. Let $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ be a data exchange setting.

1. Let q be a k-ary query over the target schema $\mathbf{T}$. *Computing the certain answers of* q is the following decision problem: given a source instance I over $\mathbf{S}$ and a k-tuple t of constants from I, is it the case that $t \in$ *certain*(q, I)?
2. Let q be a Boolean query over the target schema $\mathbf{T}$. *Computing the certain answers of* q is the following decision problem: given a source instance I over $\mathbf{S}$, is it the case that *certain*$(q, I) = \underline{\mathsf{true}}$?
3. Let $\mathcal{C}$ be a complexity class and $\mathcal{Q}$ a class of queries over the target schema $\mathbf{T}$. We say that *computing the certain answers of queries in* $\mathcal{Q}$ *is in* $\mathcal{C}$ if for every query $q \in \mathcal{Q}$, computing the certain answers of q is in $\mathcal{C}$. We say that *computing the certain answers of queries in* $\mathcal{Q}$ *is* $\mathcal{C}$*-complete* if it is in $\mathcal{C}$ and there is at least one query $q \in \mathcal{Q}$ such that computing the certain answers of q is a $\mathcal{C}$-complete problem.

Thus, computing the certain answers of a k-ary query q is a decision problem. One can also consider a related function problem: given a source instance I, find the set *certain*(q, I). The latter problem has a polynomial-time reduction to the former, since there are polynomially many k-tuples from I and so we can compute the set *certain*(q, I) by going over each such k-tuple t and deciding whether or not $t \in$ *certain*(q, I).

5.1 Computational Complexity

Since the complexity-theoretic lower bounds and inexpressibility results presented in the sequel hold for LAV data integration systems with sound views defined by conjunctive queries, we review the definition of this type of data integration system first. A LAV data integration system with sound views defined by conjunctive queries is a special case of a data exchange setting $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ in which $\Sigma_t = \emptyset$ and each source-to-target dependency in Σ_{st} is a tgd of the form $S_i(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y})$, where S_i is some relation symbol of the source schema $\mathbf{S}$ and $\psi_{\mathbf{T}}$ is an arbitrary conjunction of atomic formulas over the target schema $\mathbf{T}$. In what follows we will refer to such a setting as a LAV *setting*.

Abiteboul and Duschka [2] showed that in the LAV setting computing the certain answers of unions of conjunctive queries with inequalities is in coNP. We extend this by showing that the same upper bound holds in the data exchange setting, provided Σ_{st} is a set of tgds and Σ_t is a union of a set of egds with a weakly acyclic set of tgds.

Theorem 3. *Consider a data exchange setting in which Σ_{st} is a set of tgds and Σ_t is a union of a set of egds with a weakly acyclic set of tgds. Let q be a union of conjunctive queries with inequalities. Then computing the certain answers of q is in* coNP.

We first note that, in the particular case when all the tgds in Σ_t are full, the theorem can be proved by using the "small model property" (essentially this argument was used in [2] for the LAV setting). However, for the more general case when the tgds in Σ_t may have existentially quantified variables, the proof is more involved. It is based on an extension of the chase, called the *disjunctive chase*; see the full version [9] for details.

Theorem 3 yields an upper bound in a fairly general data exchange setting for the complexity of computing the certain answers of unions of conjunctive queries with inequalities. It turns out, as we discuss next, that this upper bound is tight, even in fairly restricted data exchange settings. Specifically, computing certain answers for such queries is coNP-complete. Therefore no polynomial algorithm exists for computing the certain answers when the input is a universal solution, unless $\mathrm{P} = \mathrm{NP}$.

Abiteboul and Duschka [2] showed that in the LAV setting, computing certain answers of conjunctive queries with inequalities is coNP-complete. They also sketched a proof which, if correct, would establish that this problem is coNP-complete even for conjunctive queries with a single inequality. Unfortunately, the reduction is erroneous. A correct reduction cannot be produced without increasing the number of inequalities, since here we show that in the LAV setting, there is a polynomial-time algorithm for computing the certain answers of unions of conjunctive queries with at most one inequality per disjunct. Still, the result of Abiteboul and Duschka [2] is correct; in fact, the unpublished full version [3] of that paper contains a proof to the effect that in the LAV setting, computing certain answers of Boolean conjunctive queries with six inequalities is coNP-complete. A different proof of the same result can be extracted by slightly modifying the proof of Theorem 3.2 in van der Meyden [22]. Thus, the next result provides a matching lower bound for the complexity of computing the certain answers of conjunctive queries with inequalities.

Theorem 4. [2,22] *In the* LAV *setting, computing the certain answers of Boolean conjunctive queries with six or more inequalities is* coNP*-complete.*

It is an interesting technical problem to determine the minimum number of inequalities needed to give rise to a coNP-complete problem in this setting.

Conjecture 1. In the LAV setting, computing the certain answers of Boolean conjunctive queries with two inequalities is coNP-complete.

We have not been able to settle this conjecture, but have succeeded in pinpointing the complexity of computing the certain answers of *unions* of Boolean conjunctive queries with at most two inequalities per disjunct.

Theorem 5. *In the* LAV *setting, computing the certain answers of unions of Boolean conjunctive queries with at most two inequalities per disjunct is* coNP*-complete. In fact,* *this problem is* coNP-*complete even for the union of two queries the first of which is a conjunctive query and the second of which is a conjunctive query with two inequalities.*

For unions of conjunctive queries with inequalities, Theorem 5 delineates the boundary of intractability, because the next theorem asserts that computing certain answers of unions of conjunctive queries with at most one inequality per disjunct can be solved in polynomial time by an algorithm that runs on universal solutions.

Theorem 6. *Assume a data exchange setting in which Σ_{st} is a set of tgds, and Σ_t is the union of a weakly acyclic set of tgds with a set of egds. Let q be a union of conjunctive queries with at most one inequality per disjunct. Let I be a source instance and let J be an arbitrary universal solution for I. Then there is a polynomial-time algorithm with input J that computes certain(q, I).*

Corollary 3. *Assume a data exchange setting in which Σ_{st} is a set of tgds, and Σ_t is the union of a weakly acyclic set of tgds with a set of egds. Let q be a union of conjunctive queries with at most one inequality per disjunct. Then there is a polynomial-time algorithm for computing the certain answers of q.*

5.2 First-Order Inexpressibility

The following theorem shows that, in general, even for a conjunctive query q with just one inequality, there is no first-order query q^* that computes the certain answers when evaluated on the canonical universal solution. This is in strong contrast with the polynomial-time algorithm that we have seen earlier (Theorem 6). It is also in contrast with the second part of Proposition 3, where we have seen a particular example for which such a q^* exists. The proof of the theorem combines Ehrenfeucht-Fraïssé games with the chase procedure.

Theorem 7. *There is a LAV setting with source I and there is a Boolean conjunctive query q with one inequality, for which there is no first-order query q^* over the canonical universal solution J such that certain$(q, I) = q^*(J)$.*

In the full version we also show that the result holds even if we allow the first-order formula q^* to contain the predicate const that distinguishes between constants and nulls.

The next result, of particular interest to query answering in the data integration context, is a corollary to the proof of Theorem 7. It shows that for conjunctive queries with just one inequality we cannot in general find any first-order query over the *source* schema that, when evaluated on the *source* instance, computes the certain answers.

Corollary 4. *There is a LAV setting with source I and there is a Boolean conjunctive query q with one inequality, for which there is no first-order query q^* over the source schema such that certain$(q, I) = q^*(I)$.*

6 Concluding Remarks

We plan to further investigate how universal solutions can be used for query answering in data exchange. We wish to characterize when a query q can be rewritten to a first-order query q^* such that the certain answers of q can be computed by evaluating q^* on a

universal solution. We also wish to understand how well the certain answers of a query can be approximated by evaluating the same query on a universal solution and how this differs from universal solution to universal solution. Finally, an important direction is extending the notion of universal solution to cover data exchange between nested (e.g. XML) schemas.

References

1. S. Abiteboul, S. Cluet, and T. Milo. Correspondence and Translation for Heterogeneous Data. In *ICDT*, pages 351–363, 1997.
2. S. Abiteboul and O. M. Duschka. Complexity of Answering Queries Using Materialized Views. In *PODS*, pages 254–263, 1998.
3. S. Abiteboul and O. M. Duschka. Complexity of Answering Queries Using Materialized Views. Unpublished full version of [2], 2000.
4. C. Beeri and M. Y. Vardi. A Proof Procedure for Data Dependencies. *JACM*, 31(4):718–741, 1984.
5. A. Calì, D. Calvanese, G. D. Giacomo, and M. Lenzerini. Data Integration under Integrity Constraints. In *CAiSE*, pages 262–279, 2002.
6. M. A. Casanova, R. Fagin, and C. H. Papadimitriou. Inclusion Dependencies and their Interaction with Functional Dependencies. *JCSS*, 28(1):29–59, 1984.
7. S. S. Cosmadakis and P. C. Kanellakis. Functional and Inclusion Dependencies: A Graph Theoretic Approach. In *Advances in Computing Research*, volume 3, pages 163–184. 1986.
8. R. Fagin. Horn Clauses and Database Dependencies. *JACM*, 29(4):952–985, Oct. 1982.
9. R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data Exchange: Semantics and Query Answering. IBM Research Report, Nov. 2002.
10. M. Friedman, A. Y. Levy, and T. D. Millstein. Navigational Plans For Data Integration. In *AAAI*, pages 67–73, 1999.
11. A. Halevy. Answering Queries Using Views: A Survey. *VLDB Journal*, pages 270–294, 2001.
12. R. Hull and M. Yoshikawa. ILOG: Declarative Creation and Manipulation of Object Identifiers. In *VLDB*, pages 455–468, 1990.
13. M. Lenzerini. Data Integration: A Theoretical Perspective. In *PODS*, pages 233–246, 2002.
14. A. Y. Levy, A. O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering Queries Using Views. In *PODS*, pages 95–104, May 1995.
15. D. Maier, A. O. Mendelzon, and Y. Sagiv. Testing Implications of Data Dependencies. *ACM TODS*, 4(4):455–469, Dec. 1979.
16. D. Maier, J. D. Ullman, and M. Y. Vardi. On the Foundations of the Universal Relation Model. *ACM TODS*, 9(2):283–308, June 1984.
17. J. A. Makowsky. Why Horn Formulas Matter in Computer Science: Initial Structures and Generic Examples. *JCSS*, 34(2/3):266–292, April/June 1987.
18. R. J. Miller, L. M. Haas, and M. Hernández. Schema Mapping as Query Discovery. In *VLDB*, pages 77–88, 2000.
19. L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernandez, and R. Fagin. Translating Web Data. In *VLDB*, pages 598–609, 2002.
20. N. C. Shu, B. C. Housel, and V. Y. Lum. CONVERT: A High Level Translation Definition Language for Data Conversion. *Communications of the ACM*, 18(10):557–567, 1975.
21. N. C. Shu, B. C. Housel, R. W. Taylor, S. P. Ghosh, and V. Y. Lum. EXPRESS: A Data EXtraction, Processing, amd REStructuring System. *TODS*, 2(2):134–174, 1977.
22. R. van der Meyden. The Complexity of Querying Indefinite Data about Linearly Ordered Domains. *JCSS*, 54:113–135, 1997.
23. R. van der Meyden. Logical Approaches to Incomplete Information: A Survey. In *Logics for Databases and Information Systems*, pages 307–356. Kluwer, 1998.

On the Expressive Power of Data Integration Systems

Andrea Calì, 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
http://www.dis.uniroma1.it/˜*lastname*

Abstract. There are basically two approaches for designing a data integration system. In the global-as-view (GAV) approach, one maps the concepts in the global schema to views over the sources, whereas in the local-as-view (LAV) approach, one maps the sources into views over the global schema. The goal of this paper is to relate the two approaches with respect to their expressive power. The analysis is carried out in a relational database setting, where both the queries on the global schema, and the views in the mapping are conjunctive queries. We introduce the notion of query-preserving transformation, and query-reducibility between data integration systems, and we show that, when no integrity constraints are allowed in global schema, the LAV and the GAV approaches are incomparable. We then consider the addition of integrity constraints in the global schema, and present techniques for query-preserving transformations in both directions. Finally, we show that our results imply that we can always transform any system following the GLAV approach (a generalization of both LAV and GAV) into a query-preserving GAV system.

1 Introduction

Data integration is the problem of combining the data residing at different sources, and providing the user with a unified view of these data, called global (or, mediated) schema [9]. The global schema is therefore the interface by which users issue their queries to the system. The system answers the queries by accessing the appropriate sources, thus freeing the user from the knowledge on where data are, and how data are structured at the sources.

The interest in this kind of systems has been continuously growing in the last years. Many organizations face the problem of integrating data residing in several sources. Companies that build a Data Warehouse, a Data Mining, or an Enterprise Resource Planning system must address this problem. Also, integrating data in the World Wide Web is the subject of several investigations and projects nowadays. Finally, applications requiring accessing or re-engineering legacy systems must deal with the problem of integrating data stored in pre-existing sources.

The design of a data integration system is a very complex task, which comprises several different issues [10]. One of the most important aspect is the specification of the mapping between the global schema and the sources, and the use of such a specification for carrying out query processing.

Two basic approaches have been used to specify the mapping between the sources and the global schema [9, 11, 12]. The first approach, called *global-as-view* (or simply GAV), requires that the global schema is expressed in terms of the data sources. More precisely, to every element of the global schema, a view over the data sources is associated, so that its meaning is specified in terms of the data residing at the sources. The second approach, called *local-as-view* (LAV), requires the global schema to be specified independently from the sources. In turn, the sources are defined as views over the global schema. The relationships between the global schema and the sources are thus established by specifying the information content of every source in terms of a view over the global schema.

Intuitively, the GAV approach provides a method for specifying the data integration system with a more procedural flavor with respect to the LAV approach. Indeed, whereas in LAV the designer of the data integration system may concentrate on specifying the content of the source in terms of the global schema, in the GAV approach, one is forced to specify how to get the data of the global schema by queries over the sources.

A comparison of the LAV and the GAV approaches is reported in [16]. It is known that the former approach ensures an easier extensibility of the integration system, and provides a more appropriate setting for its maintenance. For example, adding a new source to the system requires only to provide the definition of the source, and does not necessarily involve changes in the global view. On the contrary, in the GAV approach, adding a new source may in principle require changing the definition of the concepts in the global schema.

It is also well known that processing queries in the LAV approach is a difficult task [15, 16, 8, 1, 7, 3, 4]. Indeed, in this approach, the only knowledge we have about the data in the global schema is through the views representing the sources, and such views provide only partial information about the data. Since the mapping associates to each source a view over the global schema, it is not immediate to infer how to use the sources in order to answer queries expressed over the global schema. Thus, extracting information from the data integration system is similar to query answering with incomplete information, which is a complex task [17]. On the other hand, query processing looks much easier in the GAV approach, where we can take advantage that the mapping directly specifies which source queries corresponds to the elements of the global schema. Indeed, in most GAV systems, query answering is based on a simple unfolding strategy.

Besides the above intuitive considerations, a deep analysis of the differences/similarities of the two approaches is still missing. The goal of this paper is to investigate on the relative expressive power of the LAV and the GAV approaches. In particular, we address the problem of checking whether a LAV system can be transformed into a GAV one, and vice-versa. Obviously, we are interested in transformations that are equivalent with respect to query answering, in the sense that we want that every query posed to the original system has the same answers when posed to the new system. To this end, we introduce the notion of query-preserving transformation, and the notion of query-reducibility between classes of data integration systems. Results on query reducibility from LAV to GAV systems may be useful, for example, to derive a procedural specification from a declarative one. Conversely, results on query reducibility from GAV to LAV may be useful to derive a declarative characterization of the content of the sources starting from a procedural specification.

We study the problem in a setting where the global schema is expressed in the relational model, and the queries used in the integration systems (both the queries on the global schema, and the queries in the mapping) are expressed in the language of conjunctive queries. We show that in such a setting none of the two transformations is possible. On the contrary, we show that the presence of integrity constraints in the global schema allows reducibility in both directions. In particular, inclusion dependencies and a simple form of equality-generating dependencies suffice for a query-preserving transformation from a LAV system into a GAV one, whereas single head full dependencies are sufficient for the other direction. Finally, we introduce the GLAV approach, where both LAV and GAV assertions are allowed in the mapping, and illustrate how to adapt the technique from LAV to GAV to devise a query-preserving transformation from GLAV to GAV.

Also, the results presented in the paper shows that techniques for answering queries under integrity constraints are relevant in data integration. In particular, several approaches to answering queries under different forms of dependencies have been proposed in the last years (see for example [14]). Our results imply that these approaches can be directly applied to query answering in LAV, GAV, and GLAV systems with inclusion dependencies. Data integration is thus a good candidate as an application for experimenting these techniques in real world settings.

The paper is organized as follows. In Section 2 we describe the formal framework we use for data integration, and we introduce the notions of query-preserving transformation, and of query-reducibility between classes of data integration systems. In Section 3 we show that in the relational model without integrity constraints, the classes of LAV and GAV systems are not mutually query-reducible. In Section 4 we present the results on query-reducibility in the case where integrity constraints are allowed in the global schema. Finally, Section 5 concludes the paper with a discussion on the GLAV approach.

2 Framework for data integration

We set up a formal framework for data integration in the relational setting. We assume that the databases involved in our framework are defined over a fixed (infinite) set $\varDelta$ of objects. A database $\mathcal{DB}$ for a relational schema $\mathcal{R}$ is a relational structure $(\varDelta^{\mathcal{DB}}, \cdot^{\mathcal{DB}})$ over $\mathcal{R}$ with $\varDelta^{\mathcal{DB}} \subseteq \varDelta$. When needed, we denote a relation r of arity n by r/n. Given a query q over $\mathcal{DB}$, we denote by $q^{\mathcal{DB}}$ the set of tuples of objects in $\varDelta^{\mathcal{DB}}$ obtained by evaluating q over $\mathcal{DB}$, i.e., the set of *answers* to q over $\mathcal{DB}$. In particular, we focus on *conjunctive queries* (CQs) with equality atoms and constants. We denote a CQ of arity n over a relational schema $\mathcal{R}$ as

$$\{ \langle X_1, \ldots, X_n \rangle \mid \varphi(X_1, \ldots, X_n, Y_1, \ldots, Y_m) \}$$

where $X_1, \ldots, X_n$ are the *distinguished variables* (not necessarily pairwise distinct), $Y_1, \ldots, Y_m$ are the existentially quantified *non-distinguished variables*, and

$\varphi(X_1, \ldots, X_n, Y_1, \ldots, Y_m)$ is a conjunction of atoms over predicate symbols in $\mathcal{R}$, involving constants, and the variables $X_1, \ldots, X_n, Y_1, \ldots, Y_m$. For a relation r/n, we write the CQ $\{\langle X_1, \ldots, X_n \rangle \mid r(X_1, \ldots, X_n)\}$ simply as r.

We consider also constraints over a relational schema. In particular, we consider inclusion dependencies, simple equality-generating dependencies, and single head full dependencies [2]. Given a relation r and a tuple $\mathbf{A}$ of distinct attributes of r, we denote the projection of r over $\mathbf{A}$ by $r[\mathbf{A}]$. Similarly, given a tuple t of r, we denote the projection of t over $\mathbf{A}$ by $t[\mathbf{A}]$. An *inclusion dependency* is a dependency of the form $r[\mathbf{A}] \subseteq r'[\mathbf{A}']$, where r and r' are two relations of a relational schema $\mathcal{R}$ and $\mathbf{A}$ and $\mathbf{A}'$ are two sequences of distinct attributes of the same arity, belonging to r and r' respectively. A database $\mathcal{DB}$ satisfies $r[\mathbf{A}] \subseteq r'[\mathbf{A}']$ if $r[\mathbf{A}]^{\mathcal{DB}} \subseteq r'[\mathbf{A}']^{\mathcal{DB}}$. A *simple equality-generating dependency* has the form $r \rightarrow A = A'$, where r is a relation of a relational schema $\mathcal{R}$, and A and A' are two distinct attributes of r. A database $\mathcal{DB}$ satisfies $r \rightarrow A = A'$ if for every tuple $t \in r^{\mathcal{DB}}$, it holds that $t[A] = t[A']$. A *single head full dependency* has the form $q \subseteq r$, where r is a relation of a relational schema $\mathcal{R}$ and q is a conjunctive query over $\mathcal{R}$ of the same arity as r. A database $\mathcal{DB}$ satisfies $q \subseteq r$ if $q^{\mathcal{DB}} \subseteq r^{\mathcal{DB}}$.

A *data integration system* $\mathcal{I}$ is a triple $\langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, where

– $\mathcal{G}$ is the *global schema*, expressed in the relational model, possibly with constraints.
– $\mathcal{S}$ is the *source schema*, also expressed in the relational model.
– $\mathcal{M}$ is the *mapping* between $\mathcal{G}$ and $\mathcal{S}$, constituted by a set of *assertions* of the form

$$q_{\mathcal{S}} \subseteq q_{\mathcal{G}}$$

where $q_{\mathcal{S}}$ and $q_{\mathcal{G}}$ are two queries of the same arity, respectively over the source schema $\mathcal{S}$ and over the global schema $\mathcal{G}$.

Intuitively, the source schema describes the schema of the data sources, which contain data, while the global schema provides a reconciled, integrated, view of the underlying sources. The assertions in the mapping establish the connection between the relations of the global schema and those of the source schema. As typical in data integration, we consider here mappings that are *sound*, i.e., the data provided by the queries over the sources satisfy the queries over the global schema, but do not necessarily characterize completely the answer of the queries over the global schema [16, 9, 7]. User queries are posed over the global schema and are answered by retrieving data from the sources, making use of the mapping.

Two basic approaches for specifying the mapping have been proposed in the literature: *global-as-view* (GAV) and *local-as-view* (LAV) [16, 9]. In the GAV approach, the mapping $\mathcal{M}$ associates to each relation g in $\mathcal{G}$ a query $\varrho_{\mathcal{S}}(g)$ over $\mathcal{S}$, i.e., a GAV mapping is a set of assertions, one for each relation g of $\mathcal{G}$, of the form

$$\varrho_{\mathcal{S}}(g) \subseteq g$$

In the LAV approach, the mapping $\mathcal{M}$ associates to each relation s in $\mathcal{S}$ a query $\varrho_{\mathcal{G}}(s)$ over $\mathcal{G}$, i.e., a LAV mapping is a set of assertions, one for each relation s of $\mathcal{S}$, of the form

$$s \subseteq \varrho_{\mathcal{G}}(s)$$

Observe that in both cases we associate to a relation (either global or local) a single query. We call *GAV (with constraints)* the class of integration systems (with constraints) with a GAV mapping. Similarly for *LAV (with constraints)*.

Given an integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, we call *source database* (for $\mathcal{I}$), a database for the source schema $\mathcal{S}$, and *global database* (for $\mathcal{I}$) a database for $\mathcal{G}$ satisfying the constraints of $\mathcal{G}$. Let $\mathcal{D}$ be a source database. A global database $\mathcal{B}$ *satisfies an assertion* $q_{\mathcal{S}} \subseteq q_{\mathcal{G}}$ *in* $\mathcal{M}$ *with respect to* $\mathcal{D}$, if $q_{\mathcal{S}}^{\mathcal{D}} \subseteq q_{\mathcal{G}}^{\mathcal{B}}$. The global database $\mathcal{B}$ is said to be *legal for* $\mathcal{I}$ *with respect to* $\mathcal{D}$, if it satisfies all assertions in the mapping $\mathcal{M}$ with respect to $\mathcal{D}$. Observe that, in general, several global databases exist that are legal for $\mathcal{I}$ with respect to $\mathcal{D}$.

Queries posed to an integration system $\mathcal{I}$ are expressed in terms of the relations in the global schema of $\mathcal{I}$. Given a source database $\mathcal{D}$ for $\mathcal{I}$, the answer $q^{\mathcal{I},\mathcal{D}}$ to a query q to $\mathcal{I}$ with respect to $\mathcal{D}$, is the set of tuples t of objects in $\mathcal{D}$ such that $t \in q^{\mathcal{B}}$ for *every* global database $\mathcal{B}$ legal for $\mathcal{I}$ with respect to $\mathcal{D}$. The set $q^{\mathcal{I},\mathcal{D}}$ is called the set of *certain answers* of q to $\mathcal{I}$ with respect to $\mathcal{D}$.

Given two integration systems $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ and $\mathcal{I}' = \langle \mathcal{G}', \mathcal{S}, \mathcal{M}' \rangle$ over the same source schema $\mathcal{S}$ and such that all relations of $\mathcal{G}$ are also relations of $\mathcal{G}'$, we say that $\mathcal{I}'$ is *query-preserving* with respect to $\mathcal{I}$, if for every query q to $\mathcal{I}$ and for every source databases $\mathcal{D}$ for $\mathcal{S}$, we have that

$$q^{\mathcal{I},\mathcal{D}} = q^{\mathcal{I}',\mathcal{D}}$$

In other words, we say that $\mathcal{I}'$ is query-preserving with respect to $\mathcal{I}$ if, given a query over the global schema of $\mathcal{I}$, the certain answers we get for the query on the two integration systems are identical.

To compare classes of integration systems, we introduce the concept of query-reducibility. A class $\mathcal{C}_1$ of integration systems is *query-reducible* to a class $\mathcal{C}_2$ of integration systems if there exist a function $f : \mathcal{C}_1 \rightarrow \mathcal{C}_2$ such that, for each $\mathcal{I}_1 \in \mathcal{C}_1$ we have that $f(\mathcal{I}_1)$ is query-preserving with respect to $\mathcal{I}_1$.

3 Comparing LAV and GAV without constraints

In this section we consider data integration systems without constraints in the global schema. We want to check whether any GAV system can be transformed into a LAV one which is query-preserving wrt it, and vice-versa. We show that both transformation are not feasible.

We begin with the transformation from LAV to GAV.

Theorem 1. *The class of LAV data integration systems is not query-reducible to the class of GAV systems.*

Proof. We prove the theorem by exhibiting a particular LAV system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, a source database $\mathcal{D}$ for $\mathcal{S}$, and a set of queries such that, for any GAV system $\mathcal{I}' = \langle \mathcal{G}', \mathcal{S}, \mathcal{M}' \rangle$, the certain answers of the queries wrt $\mathcal{D}$ differ in $\mathcal{I}$ and $\mathcal{I}'$.

The LAV system $\mathcal{I}$ is as follows. The global schema $\mathcal{G}$ is constituted by $g_1/2$ and $g_2/2$, while the source schema $\mathcal{S}$ is constituted by a single relation $s/2$. The mapping

$\mathcal{M}$ is

$$\varrho_{\mathcal{G}}(s) \ = \ \{\langle X, Y\rangle \mid g_1(X, Z) \wedge g_2(Z, Y)\}$$

By contradiction, assume there is a GAV system $\mathcal{I}' = \langle \mathcal{G}', \mathcal{S}, \mathcal{M}'\rangle$ that is query-preserving with respect to $\mathcal{I}$. Observe that, since no constraints are allowed in the global schema, the introduction of a new relation in $\mathcal{G}'$ is useless if we want to construct a system that is query-preserving wrt $\mathcal{I}$; in fact, the newly introduced predicates could not be related to g_1 and g_2. Therefore, we can assume that $\mathcal{G}' = \mathcal{G}$. It follows that the mapping $\mathcal{M}'$ has the form

$$\begin{aligned}\varrho_{\mathcal{S}}(g_1) &= \{\langle X, Y\rangle \mid \xi_1(X, Y, Z_1, \ldots, Z_{k_1}, c_1, \ldots, c_{h_1})\}\\ \varrho_{\mathcal{S}}(g_2) &= \{\langle X, Y\rangle \mid \xi_2(X, Y, W_1, \ldots, W_{k_2}, d_1, \ldots, d_{h_2})\}\end{aligned}$$

where ξ_1 and ξ_2 are conjunctions of atoms over the only relation s, $Z_1, \ldots, Z_{k_1}$ and $W_1, \ldots, W_{k_2}$ are existentially quantified variables, and $c_1, \ldots, c_{h_1}$ and $d_1, \ldots, d_{h_2}$ are constants of Δ.

We take the source database $\mathcal{D}$ to be such that $s^{\mathcal{D}} = \{\langle a, b\rangle\}$, where a and b are two constants, and we consider the following queries:

$$\begin{aligned}q_1(X, Y) &= \{\langle X, Y\rangle \mid g_1(X, Z) \wedge g_2(Z, Y)\}\\ q_2(X, Y) &= \{\langle X, Y\rangle \mid g_1(X, Y)\}\\ q_3(X, Y) &= \{\langle X, Y\rangle \mid g_2(X, Y)\}\end{aligned}$$

The certain answers of q_1, q_2, and q_3 to $\mathcal{I}$ wrt $\mathcal{D}$ are the following: $q_1^{\mathcal{I},\mathcal{D}} = \langle a, b\rangle$, $q_2^{\mathcal{I},\mathcal{D}} = \emptyset$, and $q_3^{\mathcal{I},\mathcal{D}} = \emptyset$.

If one of $\varrho_{\mathcal{S}}(g_1)^{\mathcal{D}}$ or $\varrho_{\mathcal{S}}(g_2)^{\mathcal{D}}$ is non-empty, we have that one of $q_2^{\mathcal{I}',D}$ or $q_3^{\mathcal{I}',D}$ is non-empty, and hence a contradiction. When both $\varrho_{\mathcal{S}}(g_1)^{\mathcal{D}}$ and $\varrho_{\mathcal{S}}(g_2)^{\mathcal{D}}$ are empty, we immediately obtain that $q_1^{\mathcal{I}',\mathcal{D}} = \emptyset$. Contradiction.

This result shows that the mechanism of query answering in LAV cannot be directly simulated by the corresponding mechanism in GAV, which is basically unfolding, i.e., the substitution in the user query of the global relations with their definition given by the mapping.

We now turn to the transformation from GAV to LAV.

Theorem 2. *The class of GAV data integration systems is not query-reducible to the class of LAV systems.*

Proof. We exhibit a particular GAV system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M}\rangle$ and a query such that, for any LAV system $\mathcal{I}' = \langle \mathcal{G}', \mathcal{S}, \mathcal{M}'\rangle$ we can construct a source database $\mathcal{D}$ for $\mathcal{S}$ such that the certain answers of the query to $\mathcal{I}$ and $\mathcal{I}'$ differ wrt $\mathcal{D}$.

Let $\mathcal{I}$ be as follows. The global schema $\mathcal{G}$ is constituted by a single relation $g/2$, while the source schema $\mathcal{S}$ is constituted by $s_1/2$ and $s_2/2$. The mapping $\mathcal{M}$ is

$$\varrho_{\mathcal{S}}(g) \ = \ \{\langle X, Y\rangle \mid s_1(X, Z) \wedge s_2(Z, Y)\}$$

As in the previous case, we observe that the introduction of new relations in $\mathcal{G}'$ is not significant if we want to construct a system that is query-preserving wrt $\mathcal{I}$. Hence we assume that $\mathcal{G}' = \mathcal{G}$, and the mapping $\mathcal{M}'$ has the form

$$\varrho_{\mathcal{G}}(s_1) = \{\langle X, Y\rangle \mid \eta_1(X, Y, Z_1, \ldots, Z_{k_1}, c_1, \ldots, c_{h_1})\}$$
$$\varrho_{\mathcal{G}}(s_2) = \{\langle X, Y\rangle \mid \eta_2(X, Y, W_1, \ldots, W_{k_2}, d_1, \ldots, d_{h_2})\}$$

where η_1 and η_2 are conjunctions of atoms over the only relation g, $Z_1, \ldots, Z_{k_1}, W_1, \ldots, W_{k_2}$ are existentially quantified variables, and $c_1, \ldots, c_{h_1}, d_1, \ldots, d_{h_2}$ are constants in Δ.

We define the source database $\mathcal{D}$ such that $s_1^{\mathcal{D}} = \{\langle a, b\rangle\}$ and $s_2^{\mathcal{D}} = \{\langle b, c\rangle\}$, where a, b, and c are constants, distinct from $c_1, \ldots, c_{h_1}, d_1, \ldots, d_{h_2}$. Consider the query

$$q(X, Y) \ = \ \{\langle X, Y\rangle \mid g(X, Y)\}$$

whose certain answers in $\mathcal{I}$ are $\{\langle a, c\rangle\}$.

Let $\mathcal{I}' = \langle \mathcal{G}', \mathcal{S}, \mathcal{M}'\rangle$ be a LAV system. We show that $\langle a, c\rangle \notin q^{\mathcal{I}',\mathcal{D}}$, by constructing a global database $\mathcal{B}'$ which satisfies $\mathcal{M}'$ wrt $\mathcal{D}$ and such that $\langle a, c\rangle \notin q^{\mathcal{B}'}$. We construct $g^{\mathcal{B}'}$ as follows. We associate to each variable or constant V appearing in the definition of $\varrho_{\mathcal{S}}(s_1)$ a distinct constant $\psi(V)$, such that $\psi(X) = a$, $\psi(Y) = b$, and $\psi(V) = V$ if V is a constant. Then, for each atom $g(V_1, V_2)$ appearing in $\varrho_{\mathcal{S}}(s_1)$, we add the tuple $\langle \psi(V_1), \psi(V_2)\rangle$ to $g^{\mathcal{B}'}$. We do the same for $\varrho_{\mathcal{S}}(s_2)$, with $\psi(X) = b$ and $\psi(Y) = c$. Such a construction of $g^{\mathcal{B}'}$ ensures that $\langle a, c\rangle \notin g^{\mathcal{B}'}$ (by construction) and that $\mathcal{B}'$ is legal for $\mathcal{I}'$ wrt $\mathcal{D}$, as $\langle a, b\rangle \in s_1^{\mathcal{B}'}$ and $\langle b, c\rangle \in s_2^{\mathcal{B}'}$. Therefore $\langle a, c\rangle \notin q^{\mathcal{I}',\mathcal{D}}$. This proves the claim.

This result shows that we are not able to deduce the information of a LAV mapping, which specifies the role of each source relation wrt the global schema, from the information contained in a corresponding GAV mapping, which gives direct information on how query answering may be performed.

4 Comparing LAV and GAV with constraints

We address the question of query-reducibility in the case where integrity constraints are allowed in the global schema.

One direction is almost immediate: single head full dependencies suffice for query-reducibility from GAV systems to LAV systems. Indeed, if $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M}\rangle$ is a GAV system, we define a corresponding LAV system $\mathcal{I}' = \langle \mathcal{G}', \mathcal{S}, \mathcal{M}'\rangle$ as follows. For every source relation s in $\mathcal{S}$, we define a corresponding new relation g_s in $\mathcal{G}'$, and we include in $\mathcal{M}'$ the assertion $s \subseteq \varrho_{\mathcal{G}}(g_s)$. Now, for every $\varrho_{\mathcal{S}}(g) \subseteq g$ in $\mathcal{M}$, we introduce in $\mathcal{G}'$ the single head full dependency $\rho'_{\mathcal{S}}(g) \subseteq g$, where $\rho'_{\mathcal{S}}(g)$ denotes the conjunction obtained from $\rho_{\mathcal{S}}(g)$ by substituting every atom $s(x_1, \ldots, x_n)$ with $g_s(x_1, \ldots, x_n)$. It is easy to see that the resulting data integration system $\mathcal{I}' = \langle \mathcal{G}', \mathcal{S}, \mathcal{M}'\rangle$ is a LAV system, and that the transformation is query-preserving. Observe also that the size of $\mathcal{I}'$ is linearly related to the size of $\mathcal{I}$.

We now turn to the question of reducing LAV systems to GAV systems. We show that, when inclusion and simple equality generating dependencies are allowed on the global schema, we can obtain from every LAV system a query-preserving GAV system. Let $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a LAV integration system. Without loss of generality, we can assume that no equality atoms appear in the conjunctive queries in the mapping $\mathcal{M}$. We define a corresponding GAV integration system $\mathcal{I}' = \langle \mathcal{G}', \mathcal{S}, \mathcal{M}' \rangle$ as follows. For technical reasons, we first rewrite all queries in the mapping $\mathcal{M}$ so that variables appear in each atom at most once, by adding suitable equalities to the body of the queries. For example, the query $\{\langle X \rangle \mid \mathsf{cites}(X, X)\}$ is rewritten as $\{\langle X \rangle \mid \mathsf{cites}(X, Y) \wedge Y = X\}$.

Then $\mathcal{I}$ is as follows:

- The set of sources $\mathcal{S}$ remains unchanged.
- The global schema $\mathcal{G}'$ is obtained from $\mathcal{G}$ by introducing:
  - a new relation $\mathit{image\_s}/n$ for each relation s/n in $\mathcal{S}$;
  - a new relation $\mathit{expand\_s}/(n+m)$ for each relation s/n in $\mathcal{S}$, where m is the number of non-distinguished variables of $\varrho_{\mathcal{G}}(s)$; we assume variables in $\varrho_{\mathcal{G}}(s)$ to be enumerated as $Z_1, \ldots, Z_{n+m}$, with $Z_1, \ldots, Z_n$ being the distinguished variables;

  and by adding the following dependencies:
  - for each relation s/n in $\mathcal{S}$ we add the inclusion dependency

$$\mathit{image\_s}[1, \ldots, n] \subseteq \mathit{expand\_s}[1, \ldots, n]$$

  - for each relation s in $\mathcal{S}$ and for each atom $g(Z_{i_1}, \ldots, Z_{i_k})$ occurring in $\varrho_{\mathcal{G}}(s)$, we add the inclusion dependency

$$\mathit{expand\_s}[i_1, \ldots, i_k] \subseteq g[1, \ldots, k]$$

  - for each relation s in $\mathcal{S}$ and for each atom $Z_i = Z_j$ occurring in $\varrho_{\mathcal{G}}(s)$, we add the simple equality generating dependency

$$\mathit{expand\_s} \rightarrow i = j$$

- The GAV mapping $\mathcal{M}'$ associates to each global relation $\mathit{image\_s}$ the query

$$\varrho_{\mathcal{S}}(\mathit{image\_s}) = s$$

  and to the remaining global relations the empty query.

It is immediate to verify the following theorem.

Theorem 3. *Let $\mathcal{I}$ be a LAV integration system, and $\mathcal{I}'$ the corresponding GAV integration system defined as above. Then $\mathcal{I}'$ can be constructed in time that is linear in the size of $\mathcal{I}$.*

We illustrate the transformation with an example.

Example 1. Consider a LAV integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ where:

- The global schema $\mathcal{G}$ is constituted by the relations $\mathsf{cites}/2$, expressing that a paper cites another paper, and $\mathsf{sameTopic}/2$, expressing that two papers are on the same topic.

– The source schema $\mathcal{S}$ is constituted by three relations: source_1, containing pairs of papers that mutually cite each other; source_2, containing pairs of papers on the same topic, each with at least one citation; and source_3, containing papers that cite themselves.
– The LAV mapping $\mathcal{M}$ between the source schema and the global schema is:

$$\begin{aligned}\varrho_{\mathcal{G}}(\mathsf{source}_1) &= \{\langle X, Y\rangle \mid \mathsf{cites}(X, Y) \wedge \mathsf{cites}(Y, X)\}\\ \varrho_{\mathcal{G}}(\mathsf{source}_2) &= \{\langle X, Y\rangle \mid \mathsf{sameTopic}(X, Y) \wedge \mathsf{cites}(X, Z) \wedge \mathsf{cites}(Y, W)\}\\ \varrho_{\mathcal{G}}(\mathsf{source}_3) &= \{\langle X\rangle \mid \mathsf{cites}(X, Y) \wedge X = Y\}\end{aligned}$$

Then the corresponding GAV integration system $\mathcal{I}' = \langle \mathcal{G}', \mathcal{S}, \mathcal{M}'\rangle$ is as follows:

– The source schema $\mathcal{S}$ remains unchanged.
– The global schema $\mathcal{G}'$ is constituted by the relations $\mathsf{cites}/2$, $\mathsf{sameTopic}/2$ as before, and the additional relations $image\_\mathsf{source}_1/2$, $image\_\mathsf{source}_2/2$, $image\_\mathsf{source}_3/1$, $expand\_\mathsf{source}_1/2$, $expand\_\mathsf{source}_2/4$, and $expand\_\mathsf{source}_3/2$. Moreover, $\mathcal{G}$ contains the following inclusion dependencies:

$$\begin{aligned}image\_\mathsf{source}_1[1,2] &\subseteq expand\_\mathsf{source}_1[1,2]\\ image\_\mathsf{source}_2[1,2] &\subseteq expand\_\mathsf{source}_2[1,2]\\ image\_\mathsf{source}_3[1] &\subseteq expand\_\mathsf{source}_3[1]\\ expand\_\mathsf{source}_1[1,2] &\subseteq \mathsf{cites}[1,2]\\ expand\_\mathsf{source}_1[2,1] &\subseteq \mathsf{cites}[1,2]\\ expand\_\mathsf{source}_2[1,3] &\subseteq \mathsf{cites}[1,2]\\ expand\_\mathsf{source}_2[2,4] &\subseteq \mathsf{cites}[1,2]\\ expand\_\mathsf{source}_3[1,2] &\subseteq \mathsf{cites}[1,2]\\ expand\_\mathsf{source}_2[1,2] &\subseteq \mathsf{sameTopic}[1,2]\\ expand\_\mathsf{source}_3 &\rightarrow 1 = 2\end{aligned}$$

– The GAV mapping $\mathcal{M}'$ is

$$\varrho_{\mathcal{S}}(image\_\mathsf{source}_i) = \mathsf{source}_i, \qquad i \in \{1, 2, 3\}$$

■

We now show that the LAV integration system $\mathcal{I}$ and the corresponding GAV integration system $\mathcal{I}'$ obtained as above are indeed query-equivalent. The proof is based on the observation that both integration systems $\mathcal{I}$ and $\mathcal{I}'$ can be captured by suitable *logic programs* (we refer to [13] for notions relative to logic programming).

We first concentrate on GAV systems. The logic program $\mathcal{P}_{\mathcal{I}'}$ associated to a GAV system $\mathcal{I}' = \langle \mathcal{G}', \mathcal{S}, \mathcal{M}'\rangle$ is defined as follows:

– For each inclusion dependency $g_1[\mathbf{A}] \subseteq g_2[\mathbf{B}]$ in $\mathcal{G}'$, where $\mathbf{A}$ and $\mathbf{B}$ are sets of attributes, we first introduce a "pseudo-rule" of the form (assuming for simplicity that the attributes in $\mathbf{A}$ and $\mathbf{B}$ are the first h ones in g_1 and g_2, respectively):

$$g_2(X_1, \ldots, X_h, X_{h+1}, \ldots, X_n) \leftarrow g_1(X_1, \ldots, X_h, Y_{h+1}, \ldots, Y_m)$$

Then, for each simple equality generating dependency in $\mathcal{G}$ of the form $g_2 \rightarrow i{=}j$, we substitute in the above pseudo-rule each occurrence of X_j with X_i. We skolemize the resulting pseudo-rule, obtaining a rule of the form

$$g_2(Z_1, \ldots, Z_k, f_{k+1}(Z_1, \ldots, Z_k), \ldots, f_n(Z_1, \ldots, Z_k)) \leftarrow g_1(Z_1, \ldots, Z_k, W_{k+1} \ldots, W_m)$$

where each f_i is a fresh Skolem function.

– For each assertion $\varrho_{\mathcal{S}}(g) \subseteq g$ in the mapping $\mathcal{M}'$, where $\varrho_{\mathcal{S}}(g) = \{\langle X_1, \ldots, X_n\rangle \mid \varphi(X_1, \ldots, X_n, Y_{n+1}, \ldots, Y_m)\}$, we have a rule of the form

$$g(X_1, \ldots, X_n) \leftarrow \varphi(X_1, \ldots, X_n, Y_{n+1}, \ldots, Y_m)$$

with the proviso that, if a simple equality generating dependency applies to g, then we have to equate the appropriate variables.

In addition, the relations in $\mathcal{S}$ can be seen as predicates that are given extensionally. That is, a source database $\mathcal{D}$ for $\mathcal{I}'$ can be seen as a finite set of ground facts in logic programming terms.

By applying results from logic programming theory [13], we can show the following lemma.

Lemma 1. *Let $\mathcal{I}'$ be a GAV integration system, $\mathcal{D}$ a source database for $\mathcal{I}'$, $\mathcal{P}_{\mathcal{I}'}$ the corresponding logic program as defined above, and M_{min} the minimal model of $\mathcal{P}_{\mathcal{I}'} \cup \mathcal{D}$. Then, given a query q over $\mathcal{G}'$, for every tuple $\langle c_1, \ldots, c_n\rangle$ of objects in $\mathcal{D}$ we have that*

$$\langle c_1, \ldots, c_n\rangle \in q^{\mathcal{I},\mathcal{D}} \qquad \textit{if and only if} \qquad \langle c_1, \ldots, c_n\rangle \in q^{M_{min}}$$

Proof (sketch). By considering the semantics of constraints in $\mathcal{G}'$, and the corresponding translation in $\mathcal{P}_{\mathcal{I}'}$, it can be shown that the certain answers of q to $\mathcal{I}'$ wrt $\mathcal{D}$ are those that are correct answers to q for the logic program $\mathcal{P}_{\mathcal{I}'} \cup \mathcal{D}$. The claim follows from the classical result in logic programming that the correct answers to a logic program are those that are true in the minimal model. □

In other words, for GAV integration systems, the tuples of constants in the certain answer to a query q are equal to those that satisfy q in the minimal model of the corresponding logic program.

Let us turn to LAV integration systems. Without loss of generality, we can assume that equality generating dependencies have been folded into queries by suitably renaming variables. Given a LAV integration system $\mathcal{I}$, we can define an associated logic program $\mathcal{P}_{\mathcal{I}}$ by introducing rules for dependencies as before, and by treating queries in the mapping as done in [5]. In particular, given the query associated to source s (for simplicity of presentation, we assume s to be a unary relation and the relations in the query to be binary)

$$\varrho_{\mathcal{G}}(s) = \{\langle X\rangle \mid g_1(X, Y_1) \wedge \cdots \wedge g_k(X, Y_k)\}$$

by applying skolemization we get

$$\varrho_{\mathcal{G}}(s) = \{\langle X\rangle \mid g_1(X, f_1(X)) \wedge \cdots \wedge g_k(X, f_k(X))\}.$$

Then, we can introduce in $\mathcal{P}_{\mathcal{I}}$ the following rules, derived from the skolemized query:

$$\begin{array}{c} g_1(X, f_1(X)) \leftarrow s(X) \\ \cdots \\ g_k(X, f_k(X)) \leftarrow s(X) \end{array}$$

Based on the results in [5], we can prove also for LAV integration systems a lemma analogous to Lemma 1.

Lemma 2. *Let $\mathcal{I}$ be a LAV integration system, $\mathcal{D}$ a source database for $\mathcal{I}$, $\mathcal{P}_\mathcal{I}$ the corresponding logic program as defined above, and M_{min} the minimal model of $\mathcal{P}_\mathcal{I} \cup \mathcal{D}$. Then, given a query q over $\mathcal{G}$, for every tuple $\langle c_1, \ldots, c_n \rangle$ of objects in $\mathcal{D}$ we have that*

$$\langle c_1, \ldots, c_n \rangle \in q^{\mathcal{I},\mathcal{D}} \qquad \textit{if and only if} \qquad \langle c_1, \ldots, c_n \rangle \in q^{M_{min}}$$

In other words, also for LAV integration systems, the tuples of constants in the certain answer to a query q are equal to those that satisfy q in the minimal model of the corresponding logic program.

With these lemmas in place we can prove our main result.

Theorem 4. *Let $\mathcal{I}$ be a LAV integration system, and $\mathcal{I}'$ the corresponding GAV integration system defined as above. Then $\mathcal{I}'$ is query-preserving wrt $\mathcal{I}$.*

Proof (sketch). Let $\mathcal{P}_\mathcal{I}$ be the logic program capturing $\mathcal{I}$ and $\mathcal{P}_{\mathcal{I}'}$ the logic program capturing $\mathcal{I}'$. Then it is possible to show that, for every source database $\mathcal{D}$ for $\mathcal{I}$ and every global relation g of the global schema $\mathcal{G}$ of $\mathcal{I}$, we have (modulo renaming of the Skolem functions) that

$$g^{M_{min}} = g^{M'_{min}}$$

where M_{min} and M'_{min} are the minimal model of $\mathcal{P}_\mathcal{I} \cup \mathcal{D}$ and of $\mathcal{P}_{\mathcal{I}'} \cup \mathcal{D}$, respectively. Hence, by considering Lemma 1 and Lemma 2, we get the claim. $\square$

5 Discussion

In the previous sections we have studied the relative expressive power of the two main approaches to data integration, namely, LAV and GAV. We have shown that, in the case where integrity constraints are not allowed in the global schema, LAV and GAV systems are not mutually query-reducible. On the other hand, the presence of integrity constraints allows us to derive query-preserving transformations in both directions.

In particular, we have demonstrated that inclusion dependencies and a simple form of equality-generating dependencies in the global schema are sufficient for transforming any LAV systems into a query-preserving GAV system whose size is linearly related to the size of the original system. Interestingly, the technique can be easily extended for transforming any GLAV system into a GAV one.

In the GLAV approach to data integration, the relationships between the global schema and the sources are established by making use of both LAV and GAV assertions [6]. More precisely, in a GLAV system, we associate a conjunctive query $q_\mathcal{G}$ over the global schema to a conjunctive query $q_\mathcal{S}$ over the source schema. Therefore, GLAV generalizes both LAV and GAV.

By exploiting the technique presented in Section 4, it is not difficult to see that any GLAV system can be transformed into a query-preserving GAV one, with the same technique presented above. The key idea is that a GLAV assertion can be transformed into a GAV assertion plus an inclusion dependency. Indeed, for each assertion

$$q_\mathcal{S} \subseteq q_\mathcal{G}$$

in the GLAV system (where the arity of both queries is n), we introduce a new relation symbol r/n in the global schema of the resulting GAV system, and we associate to r the query

$$\varrho_{\mathcal{S}}(r) \;=\; q_{\mathcal{S}}$$

plus the inclusion

$$r \;\subseteq\; q_{\mathcal{G}}$$

Now, it is immediate to verify that the above inclusion can be treated exactly with the same technique introduced in the LAV to GAV transformation, and therefore, from the GLAV system we can obtain a query-preserving GAV system whose size is linearly related to the size of the original system.

References

1. Serge Abiteboul and Oliver Duschka. Complexity of answering queries using materialized views. In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'98)*, pages 254–265, 1998.
2. Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison Wesley Publ. Co., Reading, Massachussetts, 1995.
3. 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.
4. Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. View-based query processing and constraint satisfaction. In *Proc. of the 15th IEEE Symp. on Logic in Computer Science (LICS 2000)*, pages 361–371, 2000.
5. Oliver M. Duschka and Michael R. Genesereth. Answering recursive queries using views. In *Proc. of the 16th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'97)*, pages 109–116, 1997.
6. Marc Friedman, Alon Levy, and Todd Millstein. Navigational plans for data integration. In *Proc. of the 16th Nat. Conf. on Artificial Intelligence (AAAI'99)*, pages 67–73. AAAI Press/The MIT Press, 1999.
7. 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, 1999.
8. Jarek Gryz. Query folding with inclusion dependencies. In *Proc. of the 14th IEEE Int. Conf. on Data Engineering (ICDE'98)*, pages 126–133, 1998.
9. Alon Y. Halevy. Answering queries using views: A survey. *Very Large Database J.*, 10(4):270–294, 2001.
10. Richard Hull. Managing semantic heterogeneity in databases: A theoretical perspective. In *Proc. of the 16th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'97)*, 1997.
11. Alon Y. Levy. Logic-based techniques in data integration. In Jack Minker, editor, *Logic Based Artificial Intelligence*. Kluwer Academic Publisher, 2000.
12. Chen Li and Edward Chang. Query planning with limited source capabilities. In *Proc. of the 16th IEEE Int. Conf. on Data Engineering (ICDE 2000)*, pages 401–412, 2000.
13. John W. Lloyd. *Foundations of Logic Programming (Second, Extended Edition)*. Springer, Berlin, Heidelberg, 1987.

14. Lucian Popa, Alin Deutsch, Arnaud Sahuguet, and Val Tannen. A chase too far? In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 273–284, 2000.
15. Xiaolei Qian. Query folding. In *Proc. of the 12th IEEE Int. Conf. on Data Engineering (ICDE'96)*, pages 48–55, 1996.
16. 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, 1997.
17. Ron van der Meyden. Logical approaches to incomplete information. In Jan Chomicki and Günter Saake, editors, *Logics for Databases and Information Systems*, pages 307–356. Kluwer Academic Publisher, 1998.

Query rewriting and answering under constraints in data integration systems

Andrea Calì **Domenico Lembo** **Riccardo Rosati**
Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, I-00198 Roma, Italy
{cali, lembo, rosati}@dis.uniroma1.it

Abstract

In this paper we address the problem of query answering and rewriting in global-as-view data integration systems, when key and inclusion dependencies are expressed on the global integration schema. In the case of *sound* views, we provide sound and complete rewriting techniques for a maximal class of constraints for which decidability holds. Then, we introduce a semantics which is able to cope with violations of constraints, and present a sound and complete rewriting technique for the same decidable class of constraints. Finally, we consider the decision problem of query answering and give decidability and complexity results.

1 Introduction

The task of a data integration system is to combine data residing at different sources, providing the user with a unified view of them, called *global schema*. User queries are formulated over the global schema, and the system suitably queries the sources, providing an answer to the user, who is not obliged to have any information about the sources. The problem of data integration is a crucial issue in many application domains, e.g., re-engineering legacy systems, data warehousing, data mining, data exchange.

A central aspect of query processing is the specification of the relationship between the global schema and the sources; such a specification is given in the form of a so-called *mapping*. There are basically two approaches for specifying the mapping. The first approach, called *global-as-view* (GAV), requires that a view over the sources is associated with every element of the global schema. Conversely, the second approach, called *local-as-view* (LAV), requires the sources to be defined as views over the global schema [Lenzerini, 2002; Duschka and Levy, 1997].

The global schema is a representation of the domain of interest of the data integration system: integrity constraints are expressed on such a schema to enhance its expressiveness, thus improving its capability of representing the real world.

Since sources are in general autonomous, the data provided by the sources are likely not to satisfy the constraints on the global schema. Integrity constraints have to be taken into account during query processing; otherwise, the system may return incorrect answers to the user [Fagin *et al.*, 2003; Calì *et al.*, 2002].

Another significant issue is that the sources may not provide exactly the data that satisfy the corresponding portion of the global schema; in particular, they may provide either a subset or a superset of the data satisfying the mentioned portion, and the mapping is to be considered *sound* or *complete* respectively. Mappings that are both sound and complete are called *exact*.

In this paper, we restrict our analysis to the GAV approach, which is the most used in the context of data integration. In particular, we study a relational data integration framework in which key dependencies (KDs) and inclusion dependencies (IDs) are expressed on the global schema, and the mapping is considered sound. The main contributions of this paper are the following:

1. After showing that query answering in the general case in undecidable, we provide a sound and complete query rewriting technique first for the case of IDs alone, and then for the case of KDs together with the maximal class of IDs for which the problem is decidable, called *non-key-conflicting IDs*, or simply NKCIDs (Section 3).
2. Since it is likely that data retrieved at different, autonomous sources violate the KDs, we introduce a novel semantics that is a "relaxation" of the sound semantics, and that allows minimal repairs of the data (Section 4). We then present a sound and complete query rewriting technique in the case where KDs and NKCIDs are expressed on the global schema (Section 5).
3. Finally, we present decidability and complexity results of the (decision) problem of query answering in the different cases (Section 6).

2 Formal framework for data integration

In this section we define a logical framework for data integration, based on the relational model with integrity constraints.

Syntax We consider to have an infinite, fixed alphabet Γ of constants (also called values) representing real world objects, and will take into account only databases having Γ as domain. We adopt the so-called *unique name assumption*, i.e., we assume that different constants denote different objects.

Formally, a data integration system $\mathcal{I}$ is a triple $\langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, where:

1. $\mathcal{G}$ is the *global schema* expressed in the relational model with integrity constraints. In particular, $\mathcal{G} = \langle \Psi, \Sigma_I, \Sigma_K \rangle$, where *(i)* Ψ is a set of relations, each with an associated arity that indicates the number of its attributes. The attributes of a relation r of arity n are represented by the integers $1, \ldots, n$. *(ii)* Σ_I is a set of *inclusion dependencies* (IDs), i.e. a set of assertions of the form $r_1[\mathbf{A}] \subseteq r_2[\mathbf{B}]$, where r_1, r_2 are relations in Ψ, $\mathbf{A} = A_1, \ldots, A_n$ $(n \geq 0)$ is a sequence of attributes of r_1, and $\mathbf{B} = B_1, \ldots, B_n$ is a sequence of attributes of r_2. *(iii)* Σ_K is a set of *key dependencies* (KDs), i.e., a set of assertions of the form $key(r) = \mathbf{A}$, where r is a relation in the global schema, and $\mathbf{A} = A_1, \ldots, A_n$ is a sequence of attributes of r such that for each $i \in \{1, \ldots, n-1\}$ $a_i < a_{i+1}$. We assume, without loss of generality, that the attributes in $\mathbf{A}$ are the first n attributes of r. Moreover, we assume that at most one KD is specified for each relation.
2. $\mathcal{S}$ is the *source schema*, constituted by the schemas of the various sources that are part of the data integration system. We assume that the sources are relational, and that integrity constraints expressed on $\mathcal{S}$ are satisfied data at the sources. Hence, we do not take such constraints into account in our framework.
3. $\mathcal{M}$ is the *mapping* between the global and the source schema. In our framework the mapping is defined in the GAV approach, i.e., each relation in Ψ is associated with a *view*, i.e., a query, over the sources. We indicate the mapping as a set of assertions of the form $\langle r, V \rangle$, where r is a relation and V is the associated view over the source schema. We assume that the language used to express queries in the mapping is *positive Datalog* [Abiteboul *et al.*, 1995], over the alphabet of the relation symbols in $\mathcal{S}$. A Datalog query (or program) q of arity n is a collection of rules of the form $h(\vec{\mathbf{x}}) \leftarrow conj(\vec{\mathbf{x}}, \vec{\mathbf{y}})$, where $conj(\vec{\mathbf{x}}, \vec{\mathbf{y}})$ is a set of atoms whose predicate symbols are either relation symbols in $\mathcal{S}$ or the head symbol h, and involve $\vec{\mathbf{x}} = X_1, \ldots, X_n$ and $\vec{\mathbf{y}} = Y_1, \ldots, Y_m$, where X_i and Y_j are either variables or values of Γ. We call $h(\vec{\mathbf{x}})$ the *head* of the rule, and $conj(\vec{\mathbf{x}}, \vec{\mathbf{y}})$ the *body*.

Finally, a *query* over the global schema q is a formula that is intended to extract a set of tuples of elements of Γ. The language used to express queries over $\mathcal{G}$ is *union of conjunctive queries* (UCQ) [Abiteboul *et al.*, 1995], i.e., a Datalog program such that each rule head uses the same predicate of the same arity, and only relation symbols of $\mathcal{G}$ occur in each rule body.

Semantics A *database instance* (or simply *database*) $\mathcal{C}$ for a relational schema $\mathcal{DB}$ is a set of facts of the form $r(t)$ where r is a relation of arity n in $\mathcal{DB}$ and t is an n-tuple of values of the domain alphabet Γ. We denote as $r^{\mathcal{C}}$ the set $\{t \mid r(t) \in \mathcal{C}\}$; moreover, given a Datalog query q, we denote as $q^{\mathcal{C}}$ the evaluation of q over $\mathcal{C}$, i.e., the minimal fixpoint model of q and $\mathcal{C}$ [Abiteboul *et al.*, 1995].

In order to specify the semantics of a data integration system $\mathcal{I}$, we start by considering a *source database* for $\mathcal{I}$, i.e., a database $\mathcal{D}$ for the source schema $\mathcal{S}$. Based on $\mathcal{D}$, we now specify which is the information content of the global schema $\mathcal{G}$. We call *global database* for $\mathcal{I}$ any database for $\mathcal{G}$. Formally, given a source database $\mathcal{D}$ for $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, the semantics of $\mathcal{I}$ wrt $\mathcal{D}$, denoted $sem(\mathcal{I}, \mathcal{D})$, is a set of global databases for $\mathcal{I}$, where a global database $\mathcal{B}$ is in $sem(\mathcal{I}, \mathcal{D})$ if:

1. $\mathcal{B}$ is consistent with $\mathcal{G}$, i.e., it satisfies the IDs in Σ_I and the KDs in Σ_K. More formally: *(i)* $\mathcal{B}$ satisfies an inclusion dependency $r_1[\mathbf{A}] \subseteq r_2[\mathbf{B}]$ if for each tuple t_1 in $r_1^{\mathcal{B}}$ there exists a tuple t_2 in $r_2^{\mathcal{B}}$ such that $t_1[\mathbf{A}] = t_2[\mathbf{B}]$, where $t[\mathbf{A}]$ is the projection of the tuple t over $\mathbf{A}$. If $\mathcal{B}$ satisfies all inclusion dependencies expressed on $\mathcal{G}$ we say that $\mathcal{B}$ is consistent with Σ_I; *(ii)* $\mathcal{B}$ satisfies a key dependency $key(r) = \mathbf{A}$ if for each $t_1, t_2 \in r^{\mathcal{B}}$ with $t_1 \neq t_2$ we have $t_1[\mathbf{A}] \neq t_2[\mathbf{A}]$. If $\mathcal{B}$ satisfies all key dependencies expressed on $\mathcal{G}$ we say that $\mathcal{B}$ is consistent with Σ_K.
2. $\mathcal{B}$ satisfies the mapping $\mathcal{M}$ wrt $\mathcal{D}$, i.e., it satisfies each pair $\langle r, V \rangle$ in $\mathcal{M}$ wrt $\mathcal{D}$. In particular, we say that $\mathcal{B}$ satisfies the pair $\langle r, V \rangle$ wrt $\mathcal{D}$, if all the tuples satisfying V in $\mathcal{D}$ satisfy r in $\mathcal{B}$, i.e. $V^{\mathcal{D}} \subseteq r^{\mathcal{B}}$. Note that the above definition amounts to consider any view V as *sound*, which means that the data retrieved from sources satisfy the global schema, but are not necessarily complete.

By simply evaluating each view over the source database $\mathcal{D}$, we obtain a global database, called *retrieved global database* $ret(\mathcal{I}, \mathcal{D})$, that actually satisfies the sound mapping (but that is not necessarily consistent with $\mathcal{G}$).

We give now the semantics of queries. Formally, given a source database $\mathcal{D}$ for $\mathcal{I}$ we call *answers* to a query q of arity n w.r.t. $\mathcal{I}$ and $\mathcal{D}$, the set $ans(q, \mathcal{I}, \mathcal{D})$ defined as follows: $ans(q, \mathcal{I}, \mathcal{D}) = \{\langle c_1, \ldots, c_n \rangle \mid \text{for each } \mathcal{B} \in sem(\mathcal{I}, \mathcal{D}), \langle c_1, \ldots, c_n \rangle \in q^{\mathcal{B}}\}$.

In this paper, we address the query answering problem, that is the problem of computing the set $ans(q, \mathcal{I}, \mathcal{D})$. To this aim, we make use of query rewriting techniques, i.e., we exploit the mapping $\mathcal{M}$ to reformulate the query q into another query q_r, the *rewriting*, that can be evaluated on the source database $\mathcal{D}$. We say that q_r is a *perfect rewriting* of q w.r.t. $\mathcal{I}$ if $q_r^{\mathcal{D}} = ans(q, \mathcal{I}, \mathcal{D})$ for each $\mathcal{D}$. Furthermore, with regard to decidability and complexity results, we will refer to the decision problem associated to query answering, that is, given a data integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, a source database $\mathcal{D}$, a query q of arity n over $\mathcal{G}$ and a n-tuple $\bar{t}$ of values of Γ, to establish whether $\bar{t} \in ans(q, \mathcal{I}, \mathcal{D})$.

Example 2.1 Consider a data integration system $\mathcal{I}_0 = \langle \mathcal{G}_0, \mathcal{S}_0, \mathcal{M}_0 \rangle$, referring to the context of football teams. The global schema $\mathcal{G}_0$ consists of the relation predicates $player(Pname, Pcountry, Pteam)$ and $team(Tacronym, Tname, Tleader)$, and the following constraints: $key(player) = \{Pname\}$, $key(team) = \{Tacronym\}$, $team[Tleader] \subseteq player[Pname]$.

The source schema $\mathcal{S}_0$ consists of the schemas of three sources comprising the relation s_1 of arity 4, and the relations s_2 and s_3, both of arity 3. Finally, the mapping $\mathcal{M}_0$ is defined by the two assertions

$$\begin{array}{ll} \langle player, & player(X,Y,Z) \leftarrow s_1(X,Y,Z,W) \rangle \\ \langle team, & team(X,Y,Z) \leftarrow s_2(X,Y,Z) \\ & team(X,Y,Z) \leftarrow s_3(X,Y,Z) \rangle \end{array}$$

Consider the source database $\mathcal{D}_0 = \{s_1(\textit{Totti}, \textit{ITA}, \textit{RM}, 27), s_1(\textit{Beckham}, \textit{ENG}, \textit{MU}, 28), s_2(RM, \textit{Roma}, Totti), s_3(MU, \textit{Man.Utd.}, \textit{Giggs})\}$. Then, $ret(\mathcal{I}_0, \mathcal{D}_0) = \{player(\textit{Totti}, \textit{ITA}, \textit{RM}), player(\textit{Beckham}, \textit{ENG}, \textit{MU}), team(RM, \textit{Roma}, Totti), team(MU, \textit{Man.Utd.}, \textit{Giggs})\}$. Notice that the facts in $ret(\mathcal{I}_0, \mathcal{D}_0)$ together with the foreign key constraint $team[Tleader] \subseteq player[Pname]$ impose that $Giggs$ is a player. Since the views are sound, the semantics for the integration system has to account for all the global databases that provide the country and the team of the player. Hence, $sem(\mathcal{I}_0, \mathcal{D}_0)$ contains all database instances that can be obtained by adding to $ret(\mathcal{I}_0, \mathcal{D}_0)$ (among others) at least one fact of the form $player(\textit{Giggs}, \alpha, \beta)$, where α and β are values of the domain Γ. Given the query $q(X) \leftarrow player(X, Y, Z)$, we have that $ans(q, \mathcal{I}_0, \mathcal{D}_0) = \{\textit{Totti}, \textit{Beckham}, \textit{Giggs}\}$. ■

3 Query rewriting

In this section we present algorithms for computing the perfect rewriting of a UCQ query in GAV integration systems with KDs and IDs. We first study the case in which only IDs are expressed on the global schema, then we deal with the simultaneous presence of both IDs and KDs.

Query rewriting under IDs only We start by studying query rewriting when only IDs are expressed on the global schema. To this aim, we need some preliminary definitions.

Given a conjunctive query q, we say that a variable X is *unbound* in q if it occurs only once in q, otherwise we say that X is bound in q. Notice that variables occurring in the head of the query are necessarily bound, since each of them must also occur in the query body. A *bound term* is either a bound variable or a constant.

In the following, we assume that all unbound variables in the query q are represented by the special term ξ.

Definition 3.1 Given an atom $g = s(X_1, \ldots, X_n)$ and an inclusion $I = r[i_1, \ldots, i_k] \subseteq s[j_1, \ldots, j_k]$, we say that I *is applicable to* g if, for each ℓ such that $1 \leq \ell \leq n$, if $X_\ell \neq \xi$ then there exists h such that $j_h = \ell$. Moreover, we denote with $gr(g, I)$ the atom $s(Y_1, \ldots, Y_m)$ (m is the arity of s in Ψ) where for each ℓ such that $1 \leq \ell \leq m$, $Y_\ell = X_{j_h}$ if there exists h such that $i_h = \ell$, otherwise $Y_\ell = \xi$.

Roughly speaking, an inclusion I is applicable to an atom g if the relation symbol of g corresponds to the symbol in the right-hand side of I and if all the attributes for which bound terms appear in g are propagated by the inclusion I. When I is applicable to g, $gr(g, I)$ denotes the atom obtained from g by using I as a rewriting rule whose direction is right-to-left.

Definition 3.2 Given an atom $g_1 = r(X_1, \ldots, X_n)$ and an atom $g_2 = r(Y_1, \ldots, Y_n)$, we say that g_1 *and* g_2 *unify* if for each i such that $1 \leq i \leq n$, either $X_i = Y_i$ or $X_i = \xi$ or $Y_i = \xi$. Moreover, if g_1 and g_2 unify, we denote as $U(g_1, g_2)$ the atom $r(Z_1, \ldots, Z_n)$ where, for each i, if $X_i = Y_i$ or $Y_i = \xi$ then $Z_i = X_i$, otherwise $Z_i = Y_i$.

Informally, two atoms unify if they can be made equal through a substitution of each instance of the special symbol ξ with other terms.

Below we define the algorithm ID-rewrite to compute the perfect rewriting of a union of conjunctive queries Q. Informally, the algorithm computes the closure of the set of conjunctive queries Q with respect to the following two rules:
(i) if exists a query $q \in Q$ such that $body(q)$ contains two atoms g_1 and g_2 that unify, then the algorithm computes the query $reduce(q, g_1, g_2)$, which is obtained from q by replacing g_1 and g_2 with $U(g_1, g_2)$ in the query body, and then by applying the substitution obtained in the computation of $U(g_1, g_2)$ to the whole query. Such a new query is then transformed by the function τ, which replaces with ξ each variable symbol X such that there is a single occurrence of X in q. The use of τ is necessary in order to guarantee that each unbound variable is represented by the symbol ξ. Such a query is then added to Q.
(ii) if there exists an inclusion I and a query $q \in Q$ containing an atom g such that I is applicable to g, then the algorithm adds to Q the query obtained from q by replacing g with $gr(g, I)$ in its body (denoted in the algorithm as $q[g/gr(g, I)]$). Namely, this step adds new conjunctions obtained by applying inclusion dependencies as rewriting rules (applied from right to left).

The above rules correspond respectively to steps (a) and (b) of the algorithm.

```
Algorithm ID-rewrite($\Psi, \Sigma_I, q$)
Input: relational schema $\Psi$, inclusion dependencies $\Sigma_I$,
       union of conjunctive queries $Q$
Output: perfect rewriting of $Q$
$Q' := Q$;
repeat
  $Q_{aux} := Q'$;
  for each $q \in Q_{aux}$ do
   (a) for each $g_1, g_2 \in body(q)$ do
         if $g_1$ and $g_2$ unify
         then $Q' := Q' \cup \{\tau(reduce(q, g_1, g_2))\}$;
   (b) for each $g \in body(q)$ do
         for each $I \in \Sigma_I$ do
           if $I$ is applicable to $g$
           then $Q' := Q' \cup \{ q[g/gr(g, I)] \}$
until $Q_{aux} = Q'$;
return $Q'$
```

Termination of the algorithm is immediately implied by the fact that the number of conjunctions that can be generated by the algorithm is finite, since the maximum length of a generated conjunction is equal to the maximum length of a conjunction in the body of the initial query Q, and the number of different atoms that can be generated by the algorithm is finite, since the alphabet of relation symbols used is finite (and corresponds to the relation symbols occurring in Q and in Σ_I), as well as the set of terms used (corresponding to the set of variable and constant names occurring in the query Q plus the symbol ξ).

Henceforth, we denote as Π_{ID} the UCQ returned by ID-rewrite(Ψ, Σ_I, Q). Moreover, we define the Datalog program $\Pi_{\mathcal{M}} = \{V | \langle r, V \rangle \in \mathcal{M}\}$.

Theorem 3.3 *Let $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be an integration system and let Q be a UCQ query over $\mathcal{G}$. Then, $\Pi_{ID} \cup \Pi_{\mathcal{M}}$ is a perfect rewriting of Q w.r.t. $\mathcal{I}$.*

Query rewriting under KDs and IDs Now we address the problem of query rewriting in the case where KDs and IDs are defined on the global schema. Unfortunately, KDs and IDs interact reciprocally so that the (decision) problem of query answering in this setting becomes undecidable. The following theorem is a consequence of a similar property proved in [Calì *et al.*, 2003] in the context of a single database.

Theorem 3.4 *Consider a data integration system* $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$*, with* $\mathcal{G} = \langle \Psi, \Sigma_I, \Sigma_K \rangle$*, where* Σ_I *and* Σ_K *are sets of IDs and KDs respectively. Given a source database for* $\mathcal{I}$*, a query* q *over* $\mathcal{G}$*, and a tuple* $\bar{t}$ *of values of* Γ*, the problem of calculating* $ans(q, \mathcal{I}, \mathcal{D})$ *is undecidable.*

Undecidability of calculating the certain answers to a query immediately implies undecidability of calculating the perfect rewriting [Calì *et al.*, 2003]. The problem of query answering becomes decidable if we restrict the IDs to be in a particular class, so that they do not interact with KDs.

Definition 3.5 Consider a data integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, with $\mathcal{G} = \langle \Psi, \Sigma_I, \Sigma_K \rangle$. An ID $r_1[\mathbf{A}_1] \subseteq r_2[\mathbf{A}_2]$ is a *non-key-conflicting ID (NKCID)* w.r.t. $\mathbf{K}$ if either: *(i)* no KD is defined on r_2; *(ii)* the KD $key(r_2) = \mathbf{K}$ is in Σ_K and $\mathbf{A}_2$ is not a strict superset of $\mathbf{K}$, i.e., $\mathbf{A}_2 \not\supset \mathbf{K}$. If all IDs in Σ_I are NKCIDs w.r.t. Σ_K, the system $\mathcal{I}$ is said *non-key-conflicting (NKC)*.

We point out that the class of NKC IDs comprises the well-known class of *foreign key dependencies*, which correspond to IDs of the form $r_1[\mathbf{A}_1] \subseteq r_2[\mathbf{A}_2]$ such that $key(r_2) = \mathbf{A}_2$.

The most important feature of a NKC data integration system is the *separation* between the IDs and the KDs; in such a case, in fact, we can take IDs into account as if the KDs were not expressed on $\mathcal{G}$.

Theorem 3.6 (Separation) *Given a NKC data integration system* $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$*, with* $\mathcal{G} = \langle \Psi, \Sigma_I, \Sigma_K \rangle$*, let* $\mathcal{I}' = \langle \mathcal{G}', \mathcal{S}, \mathcal{M} \rangle$*, with* $\mathcal{G}' = \langle \Psi, \Sigma_I, \emptyset \rangle$*, the system obtained by* $\mathcal{I}$ *by eliminating the KDs of* $\mathcal{G}$*; let* $\mathcal{D}$ *be a source database for* $\mathcal{I}$ *and* $\mathcal{I}'$*. Moreover, let* q *be a query of arity* n *over* $\mathcal{G}$ *and* $\mathcal{G}'$*, and* $\bar{t}$ *an* n*-tuple of values. We have that* $\bar{t} \notin ans(q, \mathcal{I}, \mathcal{D})$ *iff* $\mathcal{D}$ *is consistent with* Σ_K *and* $\bar{t} \notin ans(q, \mathcal{I}', \mathcal{D})$*.*

Proof (sketch). We say that $\mathcal{D}$ is consistent with Σ_K iff $ret(\mathcal{I}, \mathcal{D})$ is consistent with Σ_K. An important result, immediately derived from [Johnson and Klug, 1984], states that $ans(q, \mathcal{I}, \mathcal{D})$ is obtained by evaluating q over a (possibly infinite) database, called *chase* and denoted with $chase(ret(\mathcal{I}, \mathcal{D}))$. The chase is obtained by adding tuples to $ret(\mathcal{I}, \mathcal{D})$ in a way that the added tuples repair violations of IDs. The chase satisfies the IDs over $\mathcal{G}$, and it is a representative of all databases in $sem(\mathcal{I}, \mathcal{D})$ [Calì *et al.*, 2002].

"$\Rightarrow$"Since by hypothesis $\bar{t} \notin ans(q, \mathcal{I}, \mathcal{D})$, there exists a global database $\mathcal{B} \in sem(\mathcal{I}, \mathcal{D})$ such that $\bar{t} \notin q^{\mathcal{B}}$. *A fortiori*, $\mathcal{B}$ satisfies Σ_I, therefore $\mathcal{B} \in sem(\mathcal{I}', \mathcal{D})$, The claim follows immediately.

"$\Leftarrow$"By hypothesis, $\mathcal{D}$ is consistent with Σ_K and $\bar{t} \notin ans(q, \mathcal{I}', \mathcal{D})$. Therefore, $\bar{t} \notin q^{chase(ret(\mathcal{I}', \mathcal{D}))}$; note that $chase(ret(\mathcal{I}', \mathcal{D})) = chase(ret(\mathcal{I}, \mathcal{D}))$. It can be shown, by induction on the number of added tuples in the construction of $chase(ret(\mathcal{I}, \mathcal{D}))$, that if $\mathcal{I}$ is a NKC system, and $ret(\mathcal{I}, \mathcal{D})$ satisfies Σ_K, also $chase(ret(\mathcal{I}, \mathcal{D}))$ satisfies Σ_K. It follows that $chase(ret(\mathcal{I}, \mathcal{D}))$ is a representative for all databases in $sem(\mathcal{I}, \mathcal{D})$, and $ans(q, \mathcal{I}, \mathcal{D}) = q^{chase(ret(\mathcal{I}, \mathcal{D}))}$. The claim follows straightforwardly. □

We now go back to query rewriting. In the case of a NKC data integration system, we can apply the same technique developed for IDs alone, provided that we take into account the KDs with suitable rules. Indeed, observe that if $ret(\mathcal{I}, \mathcal{D})$ violates Σ_K, any tuple is in the answer to any query. Therefore, with regard to this issue, we first introduce a unary global relation val; the idea is that val stores all values occurring in $\mathcal{D}$. We construct a set of rules Π_{val} as follows: denoting with $\{r_1, \ldots, r_{N_r}\}$ the set of all relations in $\mathcal{G}$,

$$val(X_j) \leftarrow \quad r_i(X_1, \ldots, X_{n_i})$$

with $1 \leq i \leq N_r$ and $1 \leq j \leq n_i$. Then, consider a KD of the form $key(r) = \mathbf{K}$; without loss of generality, we suppose that r has arity m and $\mathbf{K} = \{1, \ldots, k\}$ $(k < m)$. We introduce a set of $m - k$ rules; in particular, for $k + 1 \leq i \leq m$:

$$\begin{array}{rcl} q(Y_1, \ldots, Y_n) & \leftarrow & r(X_1, \ldots, X_k, X_{k+1}, \ldots, X_m), \\ & & r(X_1, \ldots, X_k, X'_{k+1}, \ldots, X'_m), \\ & & X_i \neq X'_i, val(Y_1), \ldots, val(Y_n) \end{array}$$

We denote with Π_{KD} the set of rules introduced as described above. From the results of Section 3 and from the above observation, we derive the following result.

Theorem 3.7 *Consider a data integration system* $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$*, and a query* q *of arity* n *over* $\mathcal{G}$*. Then,* $\Pi_{\text{ID}} \cup \Pi_{\text{KD}} \cup \Pi_{val} \cup \Pi_{\mathcal{M}}$ *is a perfect rewriting of* q*.*

4 Semantics for inconsistent data sources

In the sound semantics, violations of IDs are treated "automatically"because of the nature of the semantics; instead, the violation of a single KD leads to the non-interesting case in which $sem(\mathcal{I}, \mathcal{D}) = \emptyset$.

According to a common approach in the literature on inconsistent databases [Fagin *et al.*, 1983; Lin and Mendelzon, 1998; Arenas *et al.*, 1999], we now introduce the *loosely-sound* semantics, opposed to the previous one (that we will call *strictly-sound*), in which the soundness assumption is suitably relaxed. The intuition is that in the "relaxed" semantics we are allowed to delete tuples from $ret(\mathcal{I}, \mathcal{D})$ to repair violations of KDs, as long as we "minimize" such deletions; violations of IDs are treated as in the sound semantics.

Given a source database $\mathcal{D}$, we define the following ordering $\gg_{(\mathcal{I},\mathcal{D})}$ over the global databases for $\mathcal{I}$ that are consistent with $\mathcal{G}$. Given two such global databases $\mathcal{B}_1$ and $\mathcal{B}_2$, we write $\mathcal{B}_1 \gg_{(\mathcal{I},\mathcal{D})} \mathcal{B}_2$, iff $\mathcal{B}_1 \cap ret(\mathcal{I}, \mathcal{D}) \supset \mathcal{B}_2 \cap ret(\mathcal{I}, \mathcal{D})$. That is, the portion of $ret(\mathcal{I}, \mathcal{D})$ contained in the global database is greater in $\mathcal{B}_1$ than in $\mathcal{B}_2$, i.e., $\mathcal{B}_1$ approximates the sound mapping better than $\mathcal{B}_2$.

We call *maximal* w.r.t. $(\mathcal{I}, \mathcal{D})$ a global database $\mathcal{B}$ for $\mathcal{I}$ consistent with $\mathcal{G}$, such that there exists no global database $\mathcal{B}'$ consistent with $\mathcal{G}$ such that $\mathcal{B}' \gg_{(\mathcal{I},\mathcal{D})} \mathcal{B}$. Based on this notion, we define the loosely-sound semantics sem_l as follows: $sem_l(\mathcal{I}, \mathcal{D}) = \{\mathcal{B} \mid \mathcal{B} \text{ is consistent with } \mathcal{G} \text{ and } \mathcal{B} \text{ is maximal w.r.t. } (\mathcal{I}, \mathcal{D})\}$. Finally, we denote with $ans_l(q, \mathcal{I}, \mathcal{D})$ the set of answers to queries under the loosely-sound semantics.

Example 2.1 (cont.) Consider now the source database $\mathcal{D}'$ obtained by adding to $\mathcal{D}_0$ the fact $s_2(RM, \mathit{Roma}, \mathit{Beckham})$. Then, $ret(\mathcal{I}_0, \mathcal{D}') = ret(\mathcal{I}_0, \mathcal{D}_0) \cup \{team(RM, \mathit{Roma}, \mathit{Beckham})\}$ We have now that the tuples in $ret(\mathcal{I}_0, \mathcal{D}')$ violate also the KD $key(team) = \{\mathit{Tacronym}\}$; hence, $sem_l(\mathcal{I}_0, \mathcal{D}')$ contains the databases of the forms $\{player(\mathit{Totti}, \mathit{ITA}, \mathit{RM})$, $player(\mathit{Beckham}, \mathit{ENG}, \mathit{MU})$, $team(MU, \mathit{Man.Utd.}, \mathit{Giggs})$, $team(RM, \mathit{Roma}, \mathit{Totti})$, $player(\mathit{Giggs}, \alpha, \beta)\}$ and $\{player(\mathit{Totti}, \mathit{ITA}, \mathit{RM})$, $player(\mathit{Beckham}, \mathit{ENG}, \mathit{MU})$, $team(MU, \mathit{Man.Utd.}, \mathit{Giggs})$, $team(RM, \mathit{Roma}, \mathit{Beckham})$, $player(\mathit{Giggs}, \alpha, \beta)\}$, for each $\alpha, \beta \in \Gamma$.
Notice that for the query $q(X) \leftarrow player(X, Y, Z)$ $ans_l(q, \mathcal{I}_0, \mathcal{D}') = ans(q, \mathcal{I}_0, \mathcal{D})$. On the other hand, given the query $q'(X, Z) \leftarrow team(X, Y, Z)$ we have that $ans_l(q', \mathcal{I}_0, \mathcal{D}') = \{\langle MU, \mathit{Giggs}\rangle\}$ whereas $ans(q', \mathcal{I}_0, \mathcal{D}) = \{\langle MU, \mathit{Giggs}\rangle, \langle RM, \mathit{Totti}\rangle\}$. ■

It is immediate to verify that, if $sem(\mathcal{I}, \mathcal{D}) \neq \emptyset$, then $sem_l(\mathcal{I}, \mathcal{D}) = sem(\mathcal{I}, D)$, i.e., if there exists a global database that satisfies both the constraints on $\mathcal{G}$ and the mapping assertions in $\mathcal{M}$ w.r.t. a source database $\mathcal{D}$, then the strictly-sound and the loosely-sound semantics coincide.

5 Query rewriting in loosely-sound semantics

We now address the problem of computing answers to a query under the loosely-sound semantics. Specifically, we present a rewriting technique to compute answers to queries posed to NKC systems under the loosely-sound semantics.

Our method relies on Theorem 3.6 stating that for NKC systems it is possible to "separately" deal with inclusion and key dependencies: actually, for the first ones we exploit the algorithm ID-rewrite(Ψ, Σ_I, Q) presented in Section 3, whereas for the second ones we make use of Datalog$^\neg$ under stable model semantics , a well-known extension of Datalog that allows for using negation in the body of program rules [Kolaitis and Papadimitriou, 1991].

More specifically, we define a Datalog$^\neg$ program Π_{lKD} that allows us to compute the maximal subsets of $ret(\mathcal{I}, \mathcal{D})$ that are consistent with Σ_K. Π_{lKD} is obtained by taking, for each relation $r \in \mathcal{G}$, the rules

$$\begin{array}{lcl} r(\vec{\mathbf{x}}, \vec{\mathbf{y}}) & \leftarrow & r_{\mathcal{D}}(\vec{\mathbf{x}}, \vec{\mathbf{y}}) \ , \ not \ \overline{r}(\vec{\mathbf{x}}, \vec{\mathbf{y}}) \\ \overline{r}(\vec{\mathbf{x}}, \vec{\mathbf{y}}) & \leftarrow & r_{\mathcal{D}}(\vec{\mathbf{x}}, \vec{\mathbf{y}}) \ , \ r(\vec{\mathbf{x}}, \vec{\mathbf{z}}) \ , \ Y_1 \neq Z_1 \\ & \cdots & \\ \overline{r}(\vec{\mathbf{x}}, \vec{\mathbf{y}}) & \leftarrow & r_{\mathcal{D}}(\vec{\mathbf{x}}, \vec{\mathbf{y}}) \ , \ r(\vec{\mathbf{x}}, \vec{\mathbf{z}}) \ , \ Y_m \neq Z_m \end{array}$$

where: in $r(\vec{\mathbf{x}}, \vec{\mathbf{y}})$ the variables in $\vec{\mathbf{x}}$ correspond to the attributes constituting the key of the relation r; $\vec{\mathbf{y}} = Y_1, \ldots, Y_m$ and $\vec{\mathbf{z}} = Z_1, \ldots, Z_m$.

Informally, for each relation r, Π_{lKD} contains (i) a relation $r_{\mathcal{D}}$ that represents $r^{ret(\mathcal{I},\mathcal{D})}$; (ii) a relation r that represents a subset of $r^{ret(\mathcal{I},\mathcal{D})}$ that is consistent with the KD for r; (iii) an auxiliary relation $\overline{r}$. The above rules force each stable model M of Π_{lKD} to be such that r^M is a maximal subset of tuples from $r^{ret(\mathcal{I},\mathcal{D})}$ that are consistent with the KD for r.

Then, we consider the Datalog$^\neg$ program $\Pi_{lKD} \cup \Pi_{ID} \cup \Pi_{\mathcal{MD}}$, where Π_{ID} is obtained through ID-rewrite(Ψ, Σ_I, Q), and $\Pi_{\mathcal{MD}}$ is obtained from $\Pi_{\mathcal{M}}$ by replacing each symbol r with $r_{\mathcal{D}}$.

Theorem 5.1 *Let $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M}\rangle$ be a NKC system, and Q be a UCQ of arity n over $\mathcal{G}$. Then, $\Pi_{lKD} \cup \Pi_{ID} \cup \Pi_{\mathcal{MD}}$ is a perfect rewriting of Q w.r.t. $\mathcal{I}$.*

6 Summary of complexity results

Strictly-sound semantics. Query answering is undecidable even if we allow a slightly more general class of IDs than the NKCIDs; let us define a 1-key-conflicting (1KC) data integration system as a system such that for each ID $r_1[\mathbf{A}_1] \subseteq r_2[\mathbf{A}_2]$, $\mathbf{A}_2$ can be a strict superset of $key(r_2)$ (if defined), but containing at most one attribute more than $key(r_2)$.

Theorem 6.1 *The problem of query answering in 1KC integration systems, under the strictly-sound semantics, is undecidable.*

In the strictly-sound semantics, the complexity of the decision problem of query answering is immediately derived from the rewriting of Section 3.

Theorem 6.2 *The problem of query answering in NKC integration systems, under the strictly-sound semantics, is in PTIME in data complexity.*

Proof. Trivial, since the perfect rewriting $\Pi_{ID} \cup \Pi_{KD} \cup \Pi_{val} \cup \Pi_{\mathcal{M}}$ can be evaluated in PTIME w.r.t. $\mathcal{D}$. □

Loosely-sound semantics Since, as we already said, when $sem(\mathcal{I}, \mathcal{D}) \neq \emptyset$ the strict semantics and the loose ones coincide, it is easy to see that the above properties of query answering under the strictly-sound semantics can be easily generalized.

Theorem 6.3 *The problem of query answering in 1KC integration systems, under the loosely-sound semantics, is undecidable.*

We now characterize the problem of query answering under the loosely-sound semantics in NKC systems.

Theorem 6.4 *The problem of query answering in NKC integration systems, under the loosely-sound semantics, is coNP-complete in data complexity.*

Proof (sketch). Membership in coNP follows from Theorem 5.1, and from the fact that query answering in Datalog$^\neg$ is coNP-complete in data complexity, while coNP-hardness can be easily proved by a reduction of the 3-COLORABILITY problem to our problem. □

The summary of the results we have obtained is reported in the table in Figure 1, which presents the complexity of query answering for both the strictly-sound and the loosely-sound semantics. Each row corresponds to a different class of dependencies (specified in the first two columns), while each cell of the table reports data complexity and combined complexity[1] of query answering for UCQs: for each decidable case, the complexity of the problem is complete w.r.t. the class reported. In the second column of the table, FK stands for "foreign key dependencies" (a well-known class of IDs) while GEN stands for "general IDs". We have marked with the symbol ♠ the cells corresponding either to already known results or to results immediately implied by known results.

[1]The results for combined complexity, which we cannot present in detail due to space limitations, hold under the assumption that the mapping is expressed in terms of UCQs.

KDs	**IDs**	**strictly-sound**	**loosely-sound**
no	GEN	PTIME/PSPACE$^\spadesuit$	PTIME/PSPACE$^\spadesuit$
yes	no	PTIME/NP$^\spadesuit$	coNP/$\Pi_2^p$$^\spadesuit$
yes	FK	PTIME/PSPACE	coNP/PSPACE
yes	NKC	PTIME/PSPACE	coNP/PSPACE
yes	1KC	undecidable	undecidable
yes	GEN	undecidable$^\spadesuit$	undecidable$^\spadesuit$

Figure 1: Complexity of query answering (decision problem)

7 Discussion and related work

In this paper we have presented techniques for query rewriting in data integration systems with integrity constraints. and analyzed the complexity of query answering. To this aim, we have exploited formalisms and methods both from the traditional database theory and from computational logic.

Several works in the literature address the problem of data integration under constraints on the global schema. In this respect, query rewriting under integrity constraints has been first studied in the LAV setting. In particular, [Duschka and Genesereth, 1997] presents a method for query rewriting under functional dependencies in LAV systems, which is able to compute the perfect rewriting in the case of queries and mapping expressed through conjunctive queries, and "maximally contained" rewritings in the case of recursive mappings.

Then, [Gryz, 1999] analyzes query rewriting under inclusion dependencies in LAV systems, and presents a method which is able to deal simultaneously with acyclic IDs and functional dependencies, based on an algorithm for computing the rewriting of a conjunctive query in a database with inclusion dependencies. The algorithm is based on a very interesting idea: obtaining query rewriting by computing the rewriting of each atom in a way "almost independent" of the other atoms. This can be obtained if the body of the initial query q is preliminarly "minimized". However, we have found out that Gryz's algorithm does not actually compute the perfect rewriting, in the sense that some conjunctions of the perfect rewriting are missing. Our algorithm ID-rewrite presented in Section 3 is certainly inspired by Gryz's main intuitions, but overcomes the above mentioned incompleteness through a new technique for generating the rewriting.

Complexity of query answering in GAV under IDs alone is immediately derived by the results in [Johnson and Klug, 1984]; in this work, the same problem is solved for a restricted class of KDs and IDs, which, however, is significantly less general than the one treated in this paper. More recently, integration under constraints in GAV systems has been addressed in [Calì *et al.*, 2002], which presents a method for query rewriting in the presence of KDs and foreign key dependencies, under a semantics analogous to our strictly-sound semantics. Thus, the method does not deal with data inconsistencies w.r.t. KDs. Moreover, [Fagin *et al.*, 2003] presents an approach for dealing with integrity constraints in a GLAV setting (a generalization of LAV and GAV).

In single database settings, [Arenas *et al.*, 1999; Greco *et al.*, 2001] propose methods for consistent query answering in inconsistent databases, which are able to deal with universally quantified constraints. The semantics adopted in these works is different from the ones considered in the present paper.

Acknowledgments This research has been supported by the Projects INFOMIX (IST-2001-33570) and SEWASIE (IST-2001-34825) funded by the EU, and by the Project D2I funded by MIUR (Ministero per l'Istruzione, l'Università e la Ricerca). We thank Maurizio Lenzerini and Jarek Gryz for precious discussions about this material.

References

[Abiteboul *et al.*, 1995] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison Wesley Publ. Co., Reading, Massachussetts, 1995.

[Arenas *et al.*, 1999] Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. Consistent query answers in inconsistent databases. In *Proc. of PODS'99*, pages 68–79, 1999.

[Calì *et al.*, 2002] Andrea Calì, Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Data integration under integrity constraints. In *Proc. of CAiSE 2002*, volume 2348 of *LNCS*, pages 262–279. Springer, 2002.

[Calì *et al.*, 2003] Andrea Calì, Domenico Lembo, and Riccardo Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *Proc. of PODS 2003*, 2003. To Appear.

[Duschka and Genesereth, 1997] Oliver M. Duschka and Michael R. Genesereth. Answering recursive queries using views. In *Proc. of PODS'97*, pages 109–116, 1997.

[Duschka and Levy, 1997] Oliver M. Duschka and Alon Y. Levy. Recursive plans for information gathering. In *Proc. of IJCAI'97*, pages 778–784, 1997.

[Fagin *et al.*, 1983] Ronald Fagin, Jeffrey D. Ullman, and Moshe Y. Vardi. On the semantics of updates in databases. In *Proc. of PODS'83*, pages 352–365, 1983.

[Fagin *et al.*, 2003] Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: Semantics and query answering. In *Proc. of ICDT 2003*, pages 207–224, 2003.

[Greco *et al.*, 2001] Gianluigi Greco, Sergio Greco, and Ester Zumpano. A logic programming approach to the integration, repairing and querying of inconsistent databases. In *Proc. of ICLP'01*, volume 2237 of *LNAI*, pages 348–364. Springer, 2001.

[Gryz, 1999] Jarek Gryz. Query rewriting using views in the presence of functional and inclusion dependencies. *Information Systems*, 24(7):597–612, 1999.

[Johnson and Klug, 1984] David S. Johnson and Anthony C. Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. *J. of Computer and System Sciences*, 28(1):167–189, 1984.

[Kolaitis and Papadimitriou, 1991] Phokion G. Kolaitis and Christos H. Papadimitriou. Why not negation by fixpoint? *J. of Computer and System Sciences*, 43(1):125–144, 1991.

[Lenzerini, 2002] Maurizio Lenzerini. Data integration: A theoretical perspective. In *Proc. of PODS 2002*, pages 233–246, 2002.

[Lin and Mendelzon, 1998] Jinxin Lin and Alberto O. Mendelzon. Merging databases under constraints. *Int. J. of Cooperative Information Systems*, 7(1):55–76, 1998.

On the Decidability and Complexity of Query Answering over Inconsistent and Incomplete Databases

Andrea Calì Domenico Lembo Riccardo Rosati

Dipartimento di Informatica e Sistemistica
Università di Roma “La Sapienza”
Via Salaria 113, I-00198 Roma, Italy
{cali,lembo,rosati}@dis.uniroma1.it

ABSTRACT

In databases with integrity constraints, data may not satisfy the constraints. In this paper, we address the problem of obtaining consistent answers in such a setting, when key and inclusion dependencies are expressed on the database schema. We establish decidability and complexity results for query answering under different assumptions on data (soundness and/or completeness). In particular, after showing that the problem is in general undecidable, we identify the maximal class of inclusion dependencies under which query answering is decidable in the presence of key dependencies. Although obtained in a single database context, such results are directly applicable to data integration, where multiple information sources may provide data that are inconsistent with respect to the global view of the sources.

1. INTRODUCTION

In database applications, integrity constraints represent fundamental knowledge about the domain of interest [8, 1]. In many scenarios, data may not satisfy integrity constraints; this happens, for instance, in data integration [20, 17], where integrity constraints enrich the semantics of the global view of a set of autonomous information sources, while such constraints may be violated by data at the sources [14, 6]. In principle, the issue of dealing with integrity constraint violations is relevant in all applications involving the integration of heterogeneous information (e.g., Data Warehouses, Enterprise Resource Planning Systems, etc.). The current integration methodologies deal with this problem in a data reconciliation step, in which data are cleaned by *ad hoc* algorithms that eliminate all violations.

In the general case of a database in which data violate integrity constraints, the problem arises of how to interpret such a database. This problem has been extensively studied in several works in the area of *inconsistent databases* that have proposed a new semantic approach to the treatment of integrity constraints [15, 11, 5, 21, 3, 4, 19, 16], which we briefly illustrate in the following.

Traditionally, database theory adopts an *exact* interpretation of data, based on the *closed world assumption* [23], i.e., the interpretation of each relation r exactly corresponds to the extension of r in the database instance. In order to cope with data inconsistencies, other assumptions about data are adopted in the literature. In particular, the interpretation of each relation r can be considered either as a superset (*sound* semantics) or a subset (*complete* semantics) of the extension of r in the database instance. Although in many cases such assumptions are sufficient to guarantee the existence of a consistent interpretation of the data, in general a less strict interpretation is needed. In particular, several studies [15, 21, 3, 19] propose a *loose* semantics which selects, among all possible databases satisfying the integrity constraints, only the ones that are “as close as possible” to the actual database instance.

In this paper, we address the problem of query answering in a relational setting under the above semantics, when key and inclusion dependencies are expressed on the database schema. Specifically: *(i)* we identify the frontier between decidability and undecidability of query answering for the various semantics; *(ii)* for the decidable cases, we establish the computational complexity of the query answering problem.

A detailed summary of the results of this paper is presented in Section 6 (see Figure 1). We remark that the results we have obtained for the sound semantics extend previous studies on query containment under integrity constraints [18], while the results for the loose semantics extend known results in the field of inconsistent databases, by taking into account inclusion dependencies, which add significant complexity to the problem. In particular, the key issue in our work is that we are able to deal with infinite models for a database schema, that are to be taken into account when cyclic inclusion dependencies are present in the schema.

The paper is organized as follows. In Section 2 we recall the formal framework of relational databases with integrity constraints. In Section 3 we study decidability and complexity of query answering under sound, complete, and exact semantics. In Section 4 we introduce a loose semantics for inconsistent data. In Section 5 we prove results about decidability and complexity of query answering under the loose semantics. Section 6 concludes the paper.

2. FRAMEWORK

In this section we present the syntax and semantics of the relational model with integrity constraints. We assume that the reader is familiar with the basic notions of relational databases [1].

2.1 Syntax

We consider to have an infinite, fixed alphabet Γ of values representing real world objects, and we take into account only database instances having Γ as domain. Moreover, we assume that different values in Γ denote different objects, i.e., we adopt the so-called *unique name assumption*.

Basically, in the relational model we have to account for a set of relation symbols and a set of integrity constraints, i.e., assertions on the relation symbols that express conditions that are intended to be satisfied by database instances.

In this paper we focus our attention on inclusion and key dependencies. More formally, we indicate a *relational schema* (or simply *schema*) $\mathcal{DB}$ as a triple $\langle \mathcal{S}, \mathcal{I}, \mathcal{K} \rangle$, where:

- $\mathcal{S}$ is a set of relations, each with an associated arity that indicates the number of its attributes. The attributes of a relation r of arity n are represented by the integers $1, \dots, n$.
- $\mathcal{I}$ is a set of *inclusion dependencies* (IDs), i.e., a set of assertions of the form $r_1[\mathbf{A}] \subseteq r_2[\mathbf{B}]$, where r_1, r_2 are relations in $\mathcal{S}$, $\mathbf{A} = A_1, \dots, A_n$ is a sequence of distinct attributes of r_1, and $\mathbf{B} = B_1, \dots, B_n$ is a sequence of distinct attributes of r_2.
- $\mathcal{K}$ is a set of *key dependencies* (KDs), i.e., a set of assertions the form $key(r) = \mathbf{A}$, where r is a relation in the global schema, and $\mathbf{A} = A_1, \dots, A_n$ is a sequence of attributes of r. We assume that at most one key dependency is specified for each relation.

A *relational query* (or simply *query*) over $\mathcal{DB}$ is a formula that is intended to extract a set of tuples of values of Γ. The language used to express queries over $\mathcal{DB}$ is *union of conjunctive queries* (UCQ). A UCQ q of arity n is written in the form $q(\vec{\mathbf{x}}) \leftarrow conj_1(\vec{\mathbf{x}}, \vec{\mathbf{y}}_1) \vee \cdots \vee conj_m(\vec{\mathbf{x}}, \vec{\mathbf{y}}_m)$, where for each $i \in \{1, \dots, m\}$ $conj_i(\vec{\mathbf{x}}, \vec{\mathbf{y}}_i)$ is a conjunction of atoms whose predicate symbols are in $\mathcal{S}$, and involve $\vec{\mathbf{x}} = X_1, \dots, X_n$ and $\vec{\mathbf{y}}_i = Y_{i,1}, \dots, Y_{i,n_i}$, where X_k and Y_i, ℓ are either variables or values of Γ.

2.2 Semantics

A *database instance* (or simply *database*) $\mathcal{B}$ for a schema $\mathcal{DB}$ is a set of facts of the form $r(t)$ where r is a relation of arity n in $\mathcal{S}$ and t is an n-tuple of values from Γ. We denote as $r^{\mathcal{B}}$ the set $\{t \mid r(t) \in \mathcal{B}\}$. A database $\mathcal{B}$ for a schema $\mathcal{DB}$ is said to be *consistent* with $\mathcal{DB}$ if it satisfies all the dependencies expressed on $\mathcal{DB}$. In our framework, this means satisfying IDs in $\mathcal{I}$ and KDs in $\mathcal{K}$. More formally:

- $\mathcal{B}$ satisfies an inclusion dependency $r_1[\mathbf{A}] \subseteq r_2[\mathbf{B}]$ if for each tuple t_1 in $r_1^{\mathcal{B}}$ there exists a tuple t_2 in $r_2^{\mathcal{B}}$ such that $t_1[\mathbf{A}] = t_2[\mathbf{B}]$, where $t[\mathbf{A}]$ is the projection of the tuple t over $\mathbf{A}$. If $\mathcal{B}$ satisfies all inclusion dependencies expressed on $\mathcal{DB}$, then we say that $\mathcal{B}$ is consistent with $\mathcal{I}$;
- $\mathcal{B}$ satisfies a key dependency $key(r) = \mathbf{A}$ if for each $t_1, t_2 \in r^{\mathcal{B}}$ with $t_1 \neq t_2$ we have $t_1[\mathbf{A}] \neq t_2[\mathbf{A}]$. If $\mathcal{B}$ satisfies all key dependencies expressed on $\mathcal{DB}$ we say that $\mathcal{B}$ is consistent with $\mathcal{K}$.

Traditionally, the database theory essentially specifies a single database instance for a schema $\mathcal{DB}$. This means assuming that each relation r in S has to be considered *exact*, i.e., given a database instance $\mathcal{D}$ consistent with $\mathcal{DB}$, r is satisfied by exactly the tuples that satisfy r in $\mathcal{D}$.

On the other hand, different assumptions can be adopted for interpreting the tuples that $\mathcal{D}$ assigns to relations in $\mathcal{S}$ with respect to tuples that actually satisfy $\mathcal{DB}$. In particular, tuples in $\mathcal{D}$ can be considered a subset or a superset of the tuples that satisfy $\mathcal{DB}$, or exactly the set of tuples satisfying $\mathcal{DB}$. These interpretations give raise to three different semantics, called *sound*, *complete*, and *exact*, respectively.

Formally, given a database instance $\mathcal{D}$ for $\mathcal{DB} = \langle \mathcal{S}, \mathcal{I}, \mathcal{K} \rangle$, and an assumption x for $\mathcal{D}$, where $x \in \{s, c, e\}$ (for sound, complete, and exact semantics, respectively), the semantics of $\mathcal{DB}$ with respect to $\mathcal{D}$ and x, denoted $sem_x(\mathcal{DB}, \mathcal{D})$, is the set of database instances $\mathcal{B}$ for $\mathcal{DB}$ such that:

- $\mathcal{B}$ is consistent with $\mathcal{DB}$, i.e., it satisfies the integrity constraints in $\mathcal{I}$ and $\mathcal{K}$;
- $\mathcal{B}$ satisfies the assumptions specified on $\mathcal{D}$, i.e.:
  - $\mathcal{B} \supseteq \mathcal{D}$ when $x = s$ (sound semantics);
  - $\mathcal{B} \subseteq \mathcal{D}$ when $x = c$ (complete semantics);
  - $\mathcal{B} = \mathcal{D}$ when $x = e$ (exact semantics).

It is easy to see that, while $sem_e(\mathcal{DB}, \mathcal{D})$ contains at most a single database instance for $\mathcal{DB}$, in general $sem_s(\mathcal{DB}, \mathcal{D})$ and $sem_c(\mathcal{DB}, \mathcal{D})$ contain several databases for $\mathcal{DB}$. Furthermore, in our setting it always holds that $sem_c(\mathcal{DB}, \mathcal{D})$ is a non-empty set for each $\mathcal{DB}$ and each $\mathcal{D}$, since the empty database instance satisfies every possible set of KDs and IDs, therefore $\emptyset \in sem_c(\mathcal{DB}, \mathcal{D})$.

Finally, we give the semantics of queries. Formally, given a database instance $\mathcal{D}$ for $\mathcal{DB}$ and an assumption x for $\mathcal{D}$, where $x \in \{s, c, e\}$, we call *answers* to a query q of arity n with respect to $\mathcal{DB}$, $\mathcal{D}$ and x, the set $ans_x(q, \mathcal{DB}, \mathcal{D}) = \{\langle c_1, \dots, c_n \rangle \mid \text{for each } \mathcal{B} \in sem_x(\mathcal{DB}, \mathcal{D}), \langle c_1, \dots, c_n \rangle \in q^{\mathcal{B}} \}$, where $q^{\mathcal{B}}$ denotes the result of evaluating q over the database $\mathcal{B}$. We recall that $q^{\mathcal{B}}$ is the set of n-tuples of values of Γ $\langle c_1, \dots, c_n \rangle$, such that, when substituting each c_i for x_i, the formula $\exists \vec{\mathbf{y}}_1 . conj_1(\vec{\mathbf{x}}, \vec{\mathbf{y}}_1) \vee \cdots \vee \exists \vec{\mathbf{y}}_m . conj_m(\vec{\mathbf{x}}, \vec{\mathbf{y}}_m)$ evaluates to true in $\mathcal{B}$.

In this paper we address the decision problem associated to query answering, that is, given a database schema $\mathcal{DB}$, a database instance $\mathcal{D}$, a query q of arity n over $\mathcal{DB}$ and a n-tuple $\bar{t}$ of values of Γ, to establish whether $\bar{t} \in ans_x(q, \mathcal{DB}, \mathcal{D})$.

EXAMPLE 2.1. *Consider the database schema $\mathcal{DB} = \langle \mathcal{S}, \mathcal{I}, \mathcal{K} \rangle$ where $\mathcal{S}$ contains the two relations*[1] $\mathsf{player}(Pname, Pteam)$ *and* $\mathsf{team}(Tname, Tcity)$, *$\mathcal{I}$ contains the ID* $\mathsf{player}[Pteam] \subseteq \mathsf{team}[Tname]$, *stating that every player is enrolled in a team of a city, and $\mathcal{K} = \emptyset$. Assume to have the database instance*

$$\mathcal{D} = \{\mathsf{player}(a, b), \mathsf{player}(a, d), \mathsf{player}(e, f), \mathsf{team}(b, c)\}$$

[1]For the sake of clarity, in the example we use names to denote attributes, rather than integers.

where a, b, c, d, e, f are values from Γ. It is easy to see that $sem_e(\mathcal{DB}, \mathcal{D}) = \emptyset$, since there do not exist two tuples in team *having d and f as first component, i.e., $\mathcal{D}$ is not consistent with $\mathcal{DB}$. This in turn implies that query answering is meaningless, since every possible fact is a logical consequence of $\mathcal{DB}$ and $\mathcal{D}$: for instance, the answer to the query that asks for all team names in* team*, i.e., $q(x) \leftarrow \mathsf{team}(x, y)$, is the whole interpretation domain Γ (that is, every possible value belongs to the extension of the query).*

On the other hand,

$$sem_c(\mathcal{DB}, \mathcal{D}) = \{\{\mathsf{player}(a, b), \mathsf{team}(b, c)\}, \{\mathsf{team}(b, c)\}, \emptyset\}$$

while $sem_s(\mathcal{DB}, \mathcal{D})$ contains all databases instance that can be obtained by adding to $\mathcal{D}$ (among others) at least one fact of the form $\mathsf{team}(d, \alpha)$ and one fact of the form $\mathsf{team}(f, \beta)$, where α and β are values of the domain Γ. Notice that, since $\emptyset \in sem_c(\mathcal{DB}, \mathcal{D})$, $ans_c(q, \mathcal{DB}, \mathcal{D}) = \emptyset$, i.e., there is no answer to the query in the complete semantics, whereas $ans_s(q, \mathcal{DB}, \mathcal{D}) = \{b, d, f\}$. ■

2.3 Complexity classes

Finally, we briefly recall the complexity classes mentioned in the paper, and refer to [22] for further details. P^A (NP^A) is the class of problems that are solved in polynomial time by a deterministic (nondeterministic) Turing machine using an oracle for A, i.e., that solves in constant time any problem in A. In particular, the complexity class Σ_2^p is the class of problems that are solved in polynomial time by a nondeterministic Turing machine that uses an NP-oracle, and Π_2^p is the class of problems that are complement of a problem in Σ_2^p. Finally, PSPACE is the class of problems that can be solved by a Turing machine that uses a polynomially bounded amount of memory.

3. QUERY ANSWERING

In this section we address the problem of query answering in the presence of integrity constraints, under different assumptions on the data. We consider a database schema $\mathcal{DB} = \langle \mathcal{S}, \mathcal{I}, \mathcal{K} \rangle$, and a database instance $\mathcal{D}$ for $\mathcal{DB}$.

As illustrated in Section 2, when the data are considered complete, then the empty database always belongs to $sem_c(\mathcal{DB}, \mathcal{D})$, independently of $\mathcal{I}$ and $\mathcal{K}$; therefore, for any query q and for any tuple $\bar{t}$ we have immediately $ans_c(q, \mathcal{DB}, \mathcal{D}) = \emptyset$; hence, the answer to the decision problem is always negative. When the data are considered exact, we have two cases:

1. $\mathcal{D}$ satisfies both $\mathcal{I}$ and $\mathcal{K}$, therefore $sem_c(\mathcal{DB}, \mathcal{D}) = \{\mathcal{D}\}$ and $ans_e(q, \mathcal{DB}, \mathcal{D}) = q^{\mathcal{D}}$. So, it is immediate to establish whether $\bar{t} \in ans_e(q, \mathcal{DB}, \mathcal{D})$;

2. $\mathcal{D}$ violates either $\mathcal{I}$ or $\mathcal{K}$, therefore $sem_e(\mathcal{DB}, \mathcal{D}) = \emptyset$ and $ans_e(q, \mathcal{DB}, \mathcal{D})$ consists of all tuples of the same arity as q; the answer to the decision problem is therefore affirmative, independently of q and $\bar{t}$.

The case where the data are considered sound is more interesting: in fact, if the inclusion dependencies in $\mathcal{I}$ are not satisfied, we may think of adding suitable facts to $\mathcal{D}$ in order to satisfy them (according to the sound semantics, we are not allowed to repair such violations by deleting facts). In this case, if $\mathcal{D}$ satisfies $\mathcal{K}$, the semantics of $\mathcal{DB}$ is constituted in general by several (possibly infinite) databases, each of which may have infinite size, since there are several ways of adding facts to $\mathcal{D}$. Query answering with sound data is therefore a difficult task, that is not decidable in all cases.

We now define a restricted class of dependencies under which query answering is decidable.

DEFINITION 3.1. *Given a database schema $\mathcal{DB} = \langle \mathcal{S}, \mathcal{I}, \mathcal{K} \rangle$, an inclusion dependency in $\mathcal{I}$ of the form $r_1[\mathbf{A}_1] \subseteq r_2[\mathbf{A}_2]$ is a* non-key-conflicting inclusion dependency *(NKCID) with respect to $\mathcal{K}$ if either: (i) no KD is defined on r_2, or (ii) the KD $key(r_2) = \mathbf{K}$ is in $\mathcal{K}$, and $\mathbf{A}_2$ is not a strict superset of $\mathbf{K}$, i.e., $\mathbf{A}_2 \not\supset \mathbf{K}$. Moreover, the schema $\mathcal{DB}$ is* non-key-conflicting *(NKC) if all the IDs in $\mathcal{I}$ are NKCIDs with respect to $\mathcal{K}$.*

Informally, a set of dependencies is NKC if no ID in $\mathcal{I}$ propagates a proper subset of the key of the relation in its right-hand side. We point out that the class of NKC IDs comprises the well-known class of *foreign key dependencies*, which corresponds to IDs of the form $r_1[\mathbf{A_1}] \subseteq r_2[\mathbf{A_2}]$ such that $key(r_2) = \mathbf{A_2}$.

We first show that, as soon as we extend the class of dependencies beyond the non-key-conflicting case, query answering is undecidable. In particular, we introduce, together with KDs, inclusion dependencies of the form $r_1[\mathbf{A}_1] \subseteq r_2[\mathbf{A}_2]$ such that, if the KD $key(r_2) = \mathbf{K}$ is in $\mathcal{K}$, $\mathbf{A}_2$ is allowed to cover $\mathbf{K}$ plus at most one attribute of r_2. We will call such IDs *1-key-conflicting IDs* (1KCIDs) with respect to $\mathcal{K}$. A *1-key-conflicting* (1KC) database schema is defined analogously to a NKC schema. We first show undecidability of implication of KDs and 1KCIDs.

THEOREM 3.2. *The problem of implication*[2] *for KDs and 1KCIDs is undecidable.*

PROOF. The proof is by reduction from the more general problem of implication of functional dependencies (FDs) and inclusion dependencies. Consider a generic instance of this problem, i.e., given a database schema $\mathcal{DB} = \langle \mathcal{S}, \mathcal{I}, \mathcal{F} \rangle$, where $\mathcal{I}$ is a set of IDs and $\mathcal{F}$ is a set of FDs, and an inclusion dependency δ. We assume that all FDs in $\mathcal{F}$ are in *normal form*, i.e. of the form $r : \mathbf{A} \rightarrow B$, where a single attribute B is in the right-hand side. We construct an *ad hoc* problem of implication of KDs and 1KCIDs, consisting of a database schema $\mathcal{DB}_1 = \langle \mathcal{S}_1, \mathcal{I}_1, \mathcal{K}_1 \rangle$, where $\mathcal{I}_1$ is a set of 1KCID with respect to $\mathcal{K}_1$, and the same dependency δ. We will show that the two problems are equivalent, i.e. $(\mathcal{I} \cup \mathcal{F}) \models \delta$ if and only if $(\mathcal{I}_1 \cup \mathcal{K}_1) \models \delta$. The dependencies $\mathcal{I}_1$ and $\mathcal{K}_1$, defined in a new database schema $\mathcal{DB}_1 = \langle \mathcal{S}_1, \mathcal{I}_1, \mathcal{K}_1 \rangle$, are constructed as follows.

- The new set of relations $\mathcal{S}_1$ includes all relations in $\mathcal{S}$ (plus those added as below).
- $\mathcal{I}_1$ includes all IDs in I (plus those added as below).
- Let φ be a FD in $\mathcal{F}$, of the form

  $$r : \mathbf{A} \rightarrow B$$

  We add to the schema an auxiliary relation r_φ of arity $|\mathbf{A}| + 1$, and we add to $\mathcal{I}_1$ the dependencies

  $$\begin{array}{rrcl} \gamma_1 : & r_\varphi[\mathbf{A}, B] & \subseteq & r[\mathbf{A}, B] \\ \gamma_2 : & r[\mathbf{A}, B] & \subseteq & r_\varphi[\mathbf{A}, B] \end{array}$$

[2] For the details about implication of database dependencies, we refer the reader to [1].

plus the key dependency

$$\varkappa: \quad key(r_\varphi) = \mathbf{A}$$

Note that all the IDs in $\mathcal{I}_2$ are 1KCIDs with respect to $\mathcal{K}_1$. The following result, whose proof is straightforward, will be used in the rest of the proof.

LEMMA 3.3. *For any database $\mathcal{B}_1$ for $\mathcal{DB}_1$, we have that $\mathcal{B}_1$ satisfies $\{\varphi, \gamma_1, \gamma_2\}$ if and only if $\mathcal{B}_1$ satisfies $\{\varkappa, \gamma_1, \gamma_2\}$.*

From this result it follows that we are able to simulate general FDs by using KDs and 1KCIDs only. Now we end the reduction by showing that $(\mathcal{I} \cup \mathcal{F}) \models \delta$ if and only if $(\mathcal{I}_1 \cup \mathcal{K}_1) \models \delta$.

"$\Rightarrow$" By contradiction, suppose $(\mathcal{I}_1 \cup \mathcal{K}_1) \not\models \delta$; then there exists a database $\mathcal{B}_1$ for $\mathcal{DB}_1$ such that $\mathcal{B}_1$ satisfies $(\mathcal{I}_1 \cup \mathcal{K}_1)$ and violates δ. Consider a database $\mathcal{B}$ for $\mathcal{DB}$ obtained from $\mathcal{B}_1$ by removing the facts associated with the relations of the form r_φ introduced in the reduction. By Lemma 3.3, $\mathcal{B}$ satisfies $(\mathcal{I} \cup \mathcal{F})$; moreover, $\mathcal{B}$ cannot satisfy δ because $\mathcal{B}$ coincides with $\mathcal{B}_1$ on the relations in $\mathcal{S}$.

"$\Leftarrow$" By contradiction, suppose $(\mathcal{I} \cup \mathcal{K}) \not\models \delta$; then there exists a database $\mathcal{B}$ for $\mathcal{DB}$ such that $\mathcal{B}$ satisfies $(\mathcal{I} \cup \mathcal{F})$ and violates δ. We construct a database $\mathcal{B}_1$ for $\mathcal{D}_1$ that coincides with $\mathcal{B}$ on the relations in $\mathcal{S}$, and such that the facts associated with the relations of the form r_φ, introduced in the reduction, are such that the dependencies of the form γ_1, γ_2 are satisfied. By Lemma 3.3, $\mathcal{B}_1$ satisfies $(\mathcal{I}_1 \cup \mathcal{K}_1)$; moreover, it cannot satisfy δ because $\mathcal{B}_1$ coincides with $\mathcal{B}$ on the relations in $\mathcal{S}$.

The reduction is clearly computable in a finite amount of time. Since implication of IDs and FDs is undecidable, the thesis follows. ☐

We now show that query answering is undecidable in the presence of KDs and 1KCIDs.

THEOREM 3.4. *Let $\mathcal{DB} = \langle \mathcal{S}, \mathcal{I}, \mathcal{K} \rangle$ be a 1KC database schema, $\mathcal{D}$ a database instance for $\mathcal{DB}$, q a query of arity n over $\mathcal{DB}$, and $\bar{t}$ a n-tuple of values of Γ. The problem of establishing whether $\bar{t} \in ans_s(q, \mathcal{DB}, \mathcal{D})$ is undecidable.*

PROOF. The proof is analogous to a proof of PSPACE-hardness of an analogous result (addressed in the context of query containment) proved by Vardi and published in [18]. We will show a counterexample in which the problem is undecidable. Let δ be the following inclusion dependency:

$$r[A_1, \ldots, A_k] \subseteq s[B_1, \ldots, B_k]$$

where r has arity n and s has arity m. Without loss of generality, δ involves the first k attributes of r and s respectively. We choose a database instance $\mathcal{D}$ for $\mathcal{DB}$ containing the single fact $r(c_1, \ldots, c_n)$. Then we consider the following boolean query:

$$q \leftarrow r(X_1, \ldots, X_n), s(X_1, \ldots, X_k, Y_{k+1}, \ldots, Y_m)$$

Note that the query q has a positive answer (i.e., $\langle\rangle \in ans_s(q, \mathcal{DB}, \mathcal{D})$) if and only if the fact $s(c_1, \ldots, c_k, d_{k+1}, \ldots, d_m)$ is in all databases in $sem_s(\mathcal{DB}, \mathcal{D})$. It is immediate to see that this is true if and only if $(\mathcal{I} \cup \mathcal{K}) \models \delta$. Since implication of 1KCIDs and KDs is undecidable, the thesis follows. ☐

As an immediate consequence of this theorem, undecidability of query answering in the presence of KDs and general IDs follows. Moreover, the problem is still undecidable if we restrict to the class of instances consistent with the key dependencies.

COROLLARY 3.5. *Let $\mathcal{DB} = \langle \mathcal{S}, \mathcal{I}, \mathcal{K} \rangle$ be a 1KC database schema, $\mathcal{D}$ a database instance for $\mathcal{DB}$ consistent with $\mathcal{K}$, q a query of arity n over $\mathcal{DB}$, and $\bar{t}$ a n-tuple of values of Γ. The problem of establishing whether $\bar{t} \in ans_s(q, \mathcal{DB}, \mathcal{D})$ is undecidable.*

PROOF. The case where $\mathcal{D}$ does not satisfy $\mathcal{K}$ is clearly decidable, since in that case the answer to the problem is always affirmative. The claim follows immediately. ☐

Now we come to query answering in the case of NKCIDs and KDs, and prove that this problem is decidable. To this aim, we need some preliminary results, presented in the milestone paper of Johnson and Klug [18], which addresses the problem of conjunctive query containment in a database $\mathcal{DB}$, in the presence of functional and inclusion dependencies. To test whether $q_1 \subseteq q_2$, we first have to "freeze" the body of q_1, considering its atoms as facts in a database instance $\mathcal{D}$, and then applying the *chase* procedure to such a database. The resulting (possibly infinite) database, denoted as $chase(\mathcal{DB}, \mathcal{D})$, is constructed by repeatedly applying, as long as it is applicable, the following rule:

> INCLUSION DEPENDENCY CHASE RULE. Suppose there is a tuple t in $r^{chase(\mathcal{DB},\mathcal{D})}$, and there is an ID $\delta \in \mathcal{I}$ of the form $r[\mathbf{X}_r] \subseteq s[\mathbf{X}_s]$. If there is no tuple t' in $s^{chase}(\mathcal{DB}, \mathcal{D})$ such that $t'[\mathbf{X}_s] = t[\mathbf{X}_r]$, then we add a new tuple t_{chase} in $s^{chase(\mathcal{DB},\mathcal{D})}$ such that $t_{chase}[\mathbf{X}_s] = t[\mathbf{X}_r]$, and for any attribute A_i of s, with $1 \leq i \leq m$ and $A_i \notin \mathbf{X}_s$, $t_{chase}[A_i]$ is a fresh value, not appearing elsewhere in the database.

Johnson and Klug have proved that $q_1 \subseteq q_2$ if and only if $q_2^{chase(\mathcal{DB},\mathcal{D})}$ is non-empty. Moreover, they have shown that, to check whether $q_2^{chase(\mathcal{DB},\mathcal{D})}$ is non-empty, only a finite portion of $chase(\mathcal{DB}, \mathcal{D})$ needs to be considered. Based on this property, they have defined a PSPACE algorithm Answer_{JK}, that checks the non-emptiness of $q_2^{chase(\mathcal{DB},\mathcal{D})}$.

In the case of query answering, we are able to exploit the technique of Johnson and Klug. More specifically, we make use of the notion of chase as specified by the following result.

LEMMA 3.6. *Consider a database schema $\mathcal{DB} = \langle \mathcal{S}, \mathcal{I}, \emptyset \rangle$ and an instance $\mathcal{D}$ for $\mathcal{DB}$; let q be a conjunctive query of arity n, and $\bar{t}$ a tuple of the same arity. We have that $\bar{t} \in ans_s(q, \mathcal{DB}, \mathcal{D})$ if and only if $\bar{t} \in q^{chase(\mathcal{DB},\mathcal{D})}$.*

PROOF (SKETCH).

"$\Rightarrow$" Since $chase(\mathcal{DB}, \mathcal{D})$ satisfies $\mathcal{I}$, it belongs to $sem_s(\mathcal{DB}, \mathcal{D})$. From the definition of $ans_s(q, \mathcal{DB}, \mathcal{D})$, it follows that $\bar{t} \in q^{chase(\mathcal{DB},\mathcal{D})}$.

"$\Leftarrow$" Analogously to [7], it can be proved by induction on the structure of $chase(\mathcal{DB}, \mathcal{D})$ that, for any database instance $\mathcal{B} \in sem_s(\mathcal{DB}, \mathcal{D})$, there exists a homomorphism μ that sends the tuples of $chase(\mathcal{DB}, \mathcal{D})$ to the tuples of $\mathcal{B}$.

By hypothesis $\bar{t} \in q^{chase(\mathcal{DB},\mathcal{D})}$, so there exists a homomorphism λ from the atoms of q to the facts of $chase(\mathcal{DB},\mathcal{D})$; the composition $\lambda \circ \mu$ witnesses that $\bar{t} \in ans_s(q,\mathcal{DB},\mathcal{D})$. □

Based on the above property, we can apply the algorithm Answer_{JK} for query answering, to check whether a tuple $\bar{t}$ belongs to $ans_s(q,\mathcal{DB},\mathcal{D})$.

We now go back to NKCIDs. The most relevant property of NKCIDs is that they do not interfere with KDs, so that we can operate with NKCIDs just as if the KDs were not defined in the schema. This property is expressed by the following result.

Theorem 3.7 (Separation). *Let $\mathcal{DB} = \langle \mathcal{S},\mathcal{I},\mathcal{K}\rangle$ be a NKC database schema, and let $\mathcal{DB}_1 = \langle \mathcal{S},\mathcal{I},\emptyset\rangle$ be the database schema obtained from $\mathcal{DB}$ by removing the KDs. Let $\mathcal{D}$ be a database instance for $\mathcal{DB}$ and $\mathcal{DB}_1$, q a query of arity n over $\mathcal{DB}$, and $\bar{t}$ a n-tuple of values of Γ. We have that $\bar{t} \notin ans_s(q,\mathcal{DB},\mathcal{D})$ iff $\mathcal{D}$ is consistent with $\mathcal{K}$ and $\bar{t} \notin ans_s(q,\mathcal{DB}_1,\mathcal{D})$.*

Proof. "⇒" By hypothesis $\bar{t} \notin ans_s(q,\mathcal{DB},\mathcal{D})$; this means that there exists a database instance $\mathcal{B}$ for $\mathcal{DB}$ that satisfies $\mathcal{I}$ and $\mathcal{K}$, and such that $\bar{t} \notin q^{\mathcal{B}}$. It is immediate to verify that $\mathcal{B}$ is also an instance for $\mathcal{DB}_1$ that satisfies $\mathcal{K}$, and therefore $\bar{t} \notin ans_s(q,\mathcal{DB}_1,\mathcal{D})$. This proves the claim.

"⇐" By hypothesis $\bar{t} \notin ans_s(q,\mathcal{DB}_1,\mathcal{D})$ and $\mathcal{D}$ satisfies $\mathcal{K}$. Before we proceed further, we need to prove the following result.

Lemma 3.8. *Let $\mathcal{DB} = \langle \mathcal{S},\mathcal{I},\mathcal{K}\rangle$ be a database schema, with KDs and NKCIDs, and $\mathcal{D}$ is an instance for $\mathcal{DB}$. Then $chase(\mathcal{DB},\mathcal{D})$ satisfies $\mathcal{I}$ and $\mathcal{K}$ if and only if $\mathcal{D}$ is consistent with $\mathcal{K}$.*

Proof. "⇒" If $\mathcal{D}$ violates any of the key dependencies, since facts are only added (and never removed) in the construction of the canonical database $chase(\mathcal{DB},\mathcal{D})$, then also $chase(\mathcal{DB},\mathcal{D})$ violates the key dependencies in $\mathcal{DB}$.

"⇐" The proof is by induction on the structure of $chase(\mathcal{DB},\mathcal{D})$. First, by hypothesis $\mathcal{D}$ is consistent with $\mathcal{K}$. For the induction step, suppose we insert in $chase(\mathcal{DB},\mathcal{D})$ a tuple t into a relation r, on which a key dependency $key(r) = \mathbf{K}$ is defined, according to the ID $s[\mathbf{A}] \subseteq r[\mathbf{B}]$. We will show that there is no violation of the key dependencies on r, by showing that t does not agree on $\mathbf{K}$ with any pre-existing tuple $\bar{t}$ in $r^{chase^*(\mathcal{DB},\mathcal{D})}$, where $chase^*(\mathcal{DB},\mathcal{D})$ is the portion of $chase(\mathcal{DB},\mathcal{D})$ constructed until the insertion of t.

According to the definition of NKCIDs, the possible cases are the following.

1. $\mathbf{B} = \mathbf{K}$. In this case we have a foreign key dependency; t and $\bar{t}$ cannot agree on $\mathbf{K}$, because in that case t wouldn't have been added.

2. $\mathbf{B} \subset \mathbf{K}$. The two tuples differ on the values of $\mathbf{B}$ (otherwise only one of the two would have been added), so they differ also on $\mathbf{K}$.

3. $\mathbf{B} \cap \mathbf{K} \neq \emptyset$ and $\mathbf{B} - \mathbf{K} \neq \emptyset$. In this case $\mathbf{B}$ partially overlaps with $key(r)$; we necessarily have $\mathbf{K} - \mathbf{B} \neq \emptyset$, otherwise $\mathbf{B}$ would be a strict superset of $\mathbf{K}$. Therefore t and $\bar{t}$ differ in the values in $\mathbf{K} - \mathbf{B}$, where t has fresh values, thus they differ *a fortiori* on $\mathbf{K}$.

4. $\mathbf{B} \cap \mathbf{K} = \emptyset$. In this case the two tuples differ in the values in $\mathbf{K}$, where t has fresh values.

□

With this result in place, we are able to extend the result of [18] to the case of NKCIDs and KDs. In fact, in this case $chase(\mathcal{DB}_1,\mathcal{D})$ (which is identical to $chase(\mathcal{DB},\mathcal{D})$ by construction) satisfies both $\mathcal{I}$ and $\mathcal{K}$. Therefore, it is also a representative of all databases in $sem_s(\mathcal{DB},\mathcal{D})$, since $sem_s(\mathcal{DB},\mathcal{D}) \subseteq sem_s(\mathcal{DB}_1,\mathcal{D})$: hence, from Lemma 3.6 it follows that $ans_s(q,\mathcal{DB},\mathcal{D}) = q^{chase(\mathcal{DB}_1,\mathcal{D})}$. The claim follows immediately. □

Based on the above theorem, we define the algorithm $\mathsf{Answer}_\mathsf{S}$, that solves query answering in the case of non-key-conflicting database schemata.

```
Algorithm $\mathsf{Answer}_\mathsf{S}(\mathcal{DB},\mathcal{D},q,t)$
Input: NKC database schema $\mathcal{DB} = \langle \mathcal{S},\mathcal{I},\mathcal{K}\rangle$,
       database instance $\mathcal{D}$,
       query $q$ of arity $n$ over $\mathcal{DB}$,
       $n$-tuple $t$ of values of $\Gamma$;
Output: true if $t \in ans_s(q,\mathcal{DB},\mathcal{D})$, false otherwise;
if $\mathcal{D}$ is not consistent with $\mathcal{K}$
then return true
else return $\mathsf{Answer}_{JK}(\mathcal{DB},\mathcal{D},q,t)$
```

To conclude the section, we present a complexity result for query answering in the presence of NKCIDs and KDs.

Theorem 3.9. *Let $\mathcal{DB} = \langle \mathcal{S},\mathcal{I},\mathcal{K}\rangle$ be a NKC database schema, $\mathcal{D}$ a database instance for $\mathcal{DB}$, q a query of arity n over $\mathcal{DB}$, and $\bar{t}$ a n-tuple of values of Γ. The problem of establishing whether $\bar{t} \in ans_s(q,\mathcal{DB},\mathcal{D})$ is PSPACE-complete with respect to combined complexity. Moreover, it is in PTIME in data complexity.*

Proof. From the results in [18], it follows directly that the problem in the case of IDs alone is PSPACE-complete; being such a case a particular case of NKCIDs and KDs (when no KD is defined, any ID is non-key-conflicting), PSPACE-hardness in our general case follows trivially.

Membership is proved by showing that the algorithm $\mathsf{Answer}_\mathsf{S}$ runs in PSPACE. Consider a database schema $\mathcal{DB} = \langle \mathcal{S},\mathcal{I},\mathcal{K}\rangle$ where $\mathcal{I}$ and $\mathcal{K}$ are sets of NKCIDs and KDs respectively. Given a database $\mathcal{D}$ for $\mathcal{DB}$, a query q of arity n over $\mathcal{G}$, and a n-tuple t of values of Γ, we want to establish whether $t \in ans_s(q,\mathcal{DB},\mathcal{D})$. Our algorithm $\mathsf{Answer}_\mathsf{S}$ proceeds as follows. The first step, clearly feasible in PTIME (and *a fortiori* in PSPACE), checks whether $\mathcal{D}$ satisfies $\mathcal{K}$. If it does not, the answer is trivial; if it does, we can apply Theorem 3.7, disregarding $\mathcal{K}$, and apply the PSPACE algorithm Answer_{JK} of [18]. All steps of $\mathsf{Answer}_\mathsf{S}$ are computable in PSPACE. Soundness and completeness of the algorithm follow immediately from Theorem 3.7 and from soundness and completeness of Answer_{JK} [18].

Membership in PTIME in data complexity follows immediately since Answer_{JK} runs in time polynomial in data complexity. □

The above complexity characterization of the problem holds even if we restrict to instances consistent with the key dependencies.

COROLLARY 3.10. *Let $\mathcal{DB} = \langle \mathcal{S}, \mathcal{I}, \mathcal{K} \rangle$ be a NKC database schema, $\mathcal{D}$ a database instance for $\mathcal{DB}$ consistent with $\mathcal{K}$, q a query of arity n over $\mathcal{DB}$, and $\bar{t}$ a n-tuple of values of Γ. The problem of establishing whether $\bar{t} \in ans_s(q, \mathcal{DB}, \mathcal{D})$ is PSPACE-complete.*

PROOF. Membership follows immediately from the general case treated in Theorem 3.9. With regard to hardness, observe that in the case where $\mathcal{D}$ does not satisfy $\mathcal{K}$ the above algorithm solves query answering in PTIME. The claim follows immediately. □

4. SEMANTICS FOR INCONSISTENT DATA

In the cases we have addressed so far, the violation of a single dependency (under the sound and exact semantics) may lead to the non-interesting case in which $sem_x(\mathcal{DB}, \mathcal{D}) = \emptyset$ ($x \in \{e, s\}$). This does not seem reasonable when the violations are due to a small set of facts. According to a common approach in the literature on inconsistent databases [15, 21, 3, 19], we now introduce less strict assumptions on data, under which we can get *consistent answers* from inconsistent database instances.

EXAMPLE 2.1 (CONTD.). As we have already shown, $sem_e(\mathcal{DB}, \mathcal{D}) = \emptyset$ since $\mathcal{D}$ does not satisfy $\mathcal{I}$, and as a consequence query processing is trivial in the exact semantics. Assume now to add the key dependency $key[\mathsf{player}] = \{Pname\}$ to $\mathcal{K}$, stating that a player cannot be enrolled in more than one team. It is easy to see that now it is also $sem_s(\mathcal{DB}, \mathcal{D}) = \emptyset$, since the facts $\mathsf{player}(a, b)$ and $\mathsf{player}(a, d)$ are not consistent with $\mathcal{K}$, and it is not possible to make $\mathcal{D}$ satisfy $\mathcal{K}$ by adding other facts to $\mathcal{D}$. On the other hand, $\mathsf{team}(b, c)$ is consistent with the dependencies in the schema, whereas the inconsistency caused by $\mathsf{player}(e, f)$ can be resolved under the sound semantics by adding a suitable fact to $\mathcal{D}$ of the form $\mathsf{player}(f, \alpha)$. Therefore, rather than the whole domain Γ, the query $q(x) \leftarrow \mathsf{team}(x, y)$ should return the answer set $\{b\}$ under the exact semantics and $\{b, f\}$ under the sound semantics. ■

A possible solution to this problem is to characterize the semantics of a database schema $\mathcal{DB} = \langle \mathcal{S}, \mathcal{I}, \mathcal{K} \rangle$ with respect to a database instance $\mathcal{D}$ in terms of those databases that *(i)* satisfy the integrity constraints on $\mathcal{DB}$, and *(ii)* approximate "at best" the satisfaction of the assumptions on $\mathcal{D}$. In other words, the integrity constraints of $\mathcal{DB}$ are considered "hard", whereas the assumptions are considered "soft".

According to the main approaches to inconsistent databases, we now propose a modified definition of the semantics that reflects the above idea. Given a possibly inconsistent database $\mathcal{D}$ for $\mathcal{DB}$, we define an ordering on the set of all databases consistent with $\mathcal{DB}$. If $\mathcal{B}_1$ and $\mathcal{B}_2$ are two such databases, we say that $\mathcal{B}_1$ is *better* than $\mathcal{B}_2$ with respect to $\mathcal{D}$, denoted as $\mathcal{B}_1 \gg_{\mathcal{D}} \mathcal{B}_2$, if:

- $\mathcal{B}_1 \cap \mathcal{D} \supset \mathcal{B}_2 \cap \mathcal{D}$ for the sound assumption
- $\mathcal{B}_1 - \mathcal{D} \subset \mathcal{B}_2 - \mathcal{D}$ for the complete assumption
- at least one of the two following conditions holds for the exact assumption:
  (i) $\mathcal{B}_1 \cap \mathcal{D} \supset \mathcal{B}_2 \cap \mathcal{D}$ and $\mathcal{B}_1 - \mathcal{D} \subseteq \mathcal{B}_2 - \mathcal{D}$;
  (ii) $\mathcal{B}_1 \cap \mathcal{D} \supseteq \mathcal{B}_2 \cap \mathcal{D}$ and $\mathcal{B}_1 - \mathcal{D} \subset \mathcal{B}_2 - \mathcal{D}$.

With this notion in place, we can modify the notion of semantics of a schema $\mathcal{DB}$ with respect to a database instance $\mathcal{D}$ and an assumption x, where $x \in \{s, c, e\}$ as usual. In order to distinguish between the semantics used so far and their modified version, in the following we refer to the former as *strict* semantics, while we call the latter *loose* semantics, and denote it with $sem_{l_x}(\mathcal{DB}, \mathcal{D})$. Namely, we call *strictly-sound*, *strictly-complete*, and *strictly-exact* the sound, complete, and exact semantics, whereas we respectively call *loosely-sound*, *loosely-complete*, and *loosely-exact* the three loose semantics. More specifically, a database $\mathcal{B}$ consistent with $\mathcal{DB}$ is in $sem_{l_x}(\mathcal{DB}, \mathcal{D})$ if $\mathcal{B}$ is maximal with respect to $\gg_{\mathcal{D}}$, i.e., for no other database $\mathcal{B}'$ consistent with $\mathcal{DB}$, we have that $\mathcal{B}' \gg_{\mathcal{D}} \mathcal{B}$. It is also immediate to verify the following lemma:

LEMMA 4.1. *Let $\mathcal{DB} = \langle \mathcal{S}, \mathcal{I}, \mathcal{K} \rangle$ be a database schema and $\mathcal{D}$ be a database instance for $\mathcal{DB}$. Then, $sem_{ls}(\mathcal{DB}, \mathcal{D}) = \bigcup_{\mathcal{D}'} sem_s(\mathcal{DB}, \mathcal{D}')$ for each $\mathcal{D}'$ maximal subset of $\mathcal{D}$ consistent with $\mathcal{K}$.*

PROOF. For each $\mathcal{B} \in sem_{ls}(\mathcal{DB}, \mathcal{D})$ consider $\mathcal{D}' = \mathcal{B} \cap \mathcal{D}$. It is easy to see that $\mathcal{D}'$ is a maximal subset of $\mathcal{D}$ consistent with $\mathcal{K}$, and that $\mathcal{B} \in sem_s(\mathcal{DB}, \mathcal{D}')$. Furthermore, for each $\mathcal{D}'$ maximal subset of $\mathcal{D}$ consistent with $\mathcal{K}$, if $\mathcal{B}' \in sem_s(\mathcal{DB}, \mathcal{D}')$, then $\mathcal{B}' \cap \mathcal{D}' = \mathcal{D}'$, hence $\mathcal{B}' \in sem_{ls}(\mathcal{DB}, \mathcal{D})$. □

With regard to answers, we indicate the set of answers to queries under the loose semantics with $ans_{l_x}(q, \mathcal{DB}, \mathcal{D})$, where $x \in \{s, c, e\}$ as usual. It is immediate to verify that, if $sem_x(\mathcal{DB}, \mathcal{D}) \neq \emptyset$ for any $x \in \{s, c, e\}$, then the strict semantics and the loose one coincide, in the sense that, for each query q $ans_x(q, \mathcal{DB}, \mathcal{D}) = ans_{l_x}(q, \mathcal{DB}, \mathcal{D})$. Consequently, since (as illustrated in Section 2) $sem_c(\mathcal{DB}, \mathcal{D}) \neq \emptyset$ for each $\mathcal{DB}$ and for each $\mathcal{D}$, it follows that the strictly-complete and the loosely-complete semantics always coincide.

Moreover, notice that the loose semantics is never empty, i.e., it always holds that $sem_{l_x}(\mathcal{DB}, \mathcal{D}) \neq \emptyset$ for any $x \in \{s, c, e\}$, even if $sem_x(\mathcal{DB}, \mathcal{D}) = \emptyset$.

EXAMPLE 2.1 (CONTD.). With regard to our ongoing example we have that:

1. $sem_{le}(\mathcal{DB}, \mathcal{D})$ contains the database $\mathcal{B}_1 = \{\mathsf{player}(a, b), \mathsf{team}(b, c)\}$, and all the databases of the form $\mathcal{B}_2 = \{\mathsf{player}(a, d), \mathsf{team}(b, c), \mathsf{team}(d, \alpha)\}$ for each $\alpha \in \Gamma$, $\mathcal{B}_3 = \{\mathsf{player}(a, b), \mathsf{player}(e, f), \mathsf{team}(b, c), \mathsf{team}(f, \alpha)\}$ for each $\alpha \in \Gamma$, and $\mathcal{B}_4 = \{\mathsf{player}(a, d), \mathsf{player}(e, f), \mathsf{team}(b, c), \mathsf{team}(d, \alpha), \mathsf{team}(f, \beta)\}$ for each $\alpha, \beta \in \Gamma$;

2. $sem_{ls}(\mathcal{DB}, \mathcal{D})$ contains the databases of the form $\mathcal{B}_3$ and $\mathcal{B}_4$, and each database consistent with $\mathcal{DB}$ that can be obtained by adding facts to a database of the form $\mathcal{B}_3$ or $\mathcal{B}_4$;

3. $sem_{lc}(\mathcal{DB}, \mathcal{D}) = sem_c(\mathcal{DB}, \mathcal{D})$.

Therefore, under the three semantics, the answers to the query $q(x) \leftarrow \mathsf{team}(x, y)$ are respectively $ans_{le}(q, \mathcal{DB}, \mathcal{D}) = \{b\}$, $ans_{ls}(q, \mathcal{DB}, \mathcal{D}) = \{b, f\}$ and $ans_{lc}(q, \mathcal{DB}, \mathcal{D}) = \emptyset$. ■

5. QUERY ANSWERING UNDER THE LOOSE SEMANTICS

In this section we analyze the problem of computing answers to queries under the loose semantics. In particular, since (as shown in Section 4) the loosely-complete and the strictly-complete semantics coincide, we study query answering under the loosely-sound semantics and the loosely-exact semantics.

5.1 Query answering under the loosely-sound semantics

We now study the query answering problem under the loosely-sound semantics. As we already said, differently from the strictly-sound semantics, given a database schema $\mathcal{DB} = \langle \mathcal{S}, \mathcal{I}, \mathcal{K} \rangle$ and a database instance $\mathcal{D}$, it always holds that $sem_{ls}(\mathcal{DB}, \mathcal{D}) \neq \emptyset$, because we are now allowed to also eliminate facts from $\mathcal{D}$ in order to satisfy integrity constraints. Notice that, while to satisfy key dependencies we are forced to delete facts from $\mathcal{D}$, inclusion dependencies must be satisfied by adding new facts, since databases in $sem_{ls}(\mathcal{DB}, \mathcal{D})$ are the ones that are "as sound as possible", thus we have to consider only databases consistent with the constraints that "minimize" elimination of facts from $\mathcal{D}$.

We first show undecidability of query answering for 1-key-conflicting database schemata.

THEOREM 5.1. *Let $\mathcal{DB} = \langle \mathcal{S}, \mathcal{I}, \mathcal{K} \rangle$ be a 1KC database schema, $\mathcal{D}$ a database instance for $\mathcal{DB}$ consistent with $\mathcal{K}$, q a query of arity n over $\mathcal{DB}$, and $\bar{t}$ a n-tuple of values from Γ. The problem of establishing whether $\bar{t} \in ans_{ls}(q, \mathcal{DB}, \mathcal{D})$ is undecidable.*

PROOF. Undecidability follows from Corollary 3.5 since, in the case in which $\mathcal{D}$ is consistent with $\mathcal{K}$, $\bar{t} \in ans_{ls}(q, \mathcal{DB}, \mathcal{D})$ iff $\bar{t} \in ans_s(q, \mathcal{DB}, \mathcal{D})$. □

As for the class of non-key-conflicting database schemata, we give a method for computing answers to a query q under the loosely-sound semantics that can be informally explained as follows: we first identify the maximal subsets of $\mathcal{D}$ that are consistent with $\mathcal{K}$, then for each such database $\mathcal{D}'$ we make use of the algorithm $\mathsf{Answer_S}$ presented in Section 3. Indeed, it can be shown that a tuple t is a consistent answer to a query q with respect to $\mathcal{DB}$ and $\mathcal{D}$, i.e., $t \in ans_{ls}(q, \mathcal{DB}, \mathcal{D})$, iff $\mathsf{Answer_S}(\mathcal{DB}, \mathcal{D}', q, t)$ returns true for each such database $\mathcal{D}'$. More specifically, we define the following algorithm:

```
Algorithm $\mathsf{Answer_{LS}}(\mathcal{DB}, \mathcal{D}, q, t)$
Input: non-key-conflicting database schema $\mathcal{DB} = \langle \mathcal{S}, \mathcal{I}, \mathcal{K} \rangle$,
       database instance $\mathcal{D}$,
       query $q$ of arity $n$ over $\mathcal{DB}$, $n$-tuple $t$ of values from $\Gamma$;
Output: true if $t \in ans_{ls}(q, \mathcal{DB}, \mathcal{D})$, false otherwise;
if there exists $\mathcal{D}_1 \subseteq \mathcal{D}$
such that
  (1) $\mathcal{D}_1$ is consistent with $\mathcal{K}$;
  (2) for each $r(\bar{t}) \in \mathcal{D} - \mathcal{D}_1$,
        $\mathcal{D}_1 \cup \{r(\bar{t})\}$ is not consistent with $\mathcal{K}$;
  (3) $\mathsf{Answer_S}(\mathcal{DB}, \mathcal{D}_1, q, t)$ returns false
then return false
else return true
```

Informally, conditions (1) and (2) together check that $\mathcal{D}_1$ is a maximal subset of $\mathcal{D}$ consistent with $\mathcal{K}$; this implies the existence of a database $\mathcal{B} \in sem_{ls}(\mathcal{DB}, \mathcal{D})$ such that $\mathcal{B} \cap \mathcal{D} = \mathcal{D}_1$. Then, condition (3) verifies that $\bar{t} \notin q^{\mathcal{B}}$.

THEOREM 5.2. *Let $\mathcal{DB} = \langle \mathcal{S}, \mathcal{I}, \mathcal{K} \rangle$ be a NKC database schema, $\mathcal{D}$ be a database instance for $\mathcal{DB}$, q be a query of arity n over $\mathcal{DB}$, and $\bar{t}$ be a n-tuple of values of Γ. Then, $\bar{t} \in ans_{ls}(q, \mathcal{DB}, \mathcal{D})$ iff $\mathsf{Answer_{LS}}(\mathcal{DB}, \mathcal{D}, q, \bar{t})$ returns true.*

PROOF. "⇒" If $\bar{t} \in ans_{ls}(q, \mathcal{DB}, \mathcal{D})$ then $\bar{t} \in q^{\mathcal{B}}$ for each $\mathcal{B} \in sem_{ls}(\mathcal{DB}, \mathcal{D})$. From Lemma 4.1 it follows that $\bar{t} \in ans_s(q, \mathcal{DB}, \mathcal{D}_1)$ for each $\mathcal{D}_1$ maximal subset of $\mathcal{D}$ consistent with $\mathcal{K}$, and from soundness and completeness of algorithm $\mathsf{Answer_S}$, it follows that $\mathsf{Answer_S}(\mathcal{DB}, \mathcal{D}_1, q, \bar{t})$ returns *true* for each such database $\mathcal{D}_1$. Hence, $\mathsf{Answer_{LS}}(\mathcal{DB}, \mathcal{D}, q, \bar{t})$ returns *true*.

"⇐" Suppose by contradiction that $\bar{t} \notin ans_{ls}(q, \mathcal{DB}, \mathcal{D})$ and $\mathsf{Answer_{LS}}(\mathcal{DB}, \mathcal{D}, q, \bar{t})$ returns *true*. This implies that for each $\mathcal{D}_1$ maximal subset of $\mathcal{D}$ consistent with $\mathcal{K}$, $\mathsf{Answer_S}(\mathcal{DB}, \mathcal{D}_1, q, \bar{t})$ returns *true*. From soundness and completeness of algorithm $\mathsf{Answer_S}$, it follows that $\bar{t} \in ans_s(q, \mathcal{DB}, \mathcal{D}_1)$ for each such database $\mathcal{D}_1$, i.e., $\bar{t} \in q^{\mathcal{B}}$ for each $\mathcal{B} \in sem_s(\mathcal{DB}, \mathcal{D}_1)$. From Lemma 4.1 it follows that $\bar{t} \in q^{\mathcal{B}}$ for each $\mathcal{B} \in sem_{ls}(\mathcal{DB}, \mathcal{D})$, but this contradicts the assumption. □

We give now the computational characterization of the problem of query answering under the loosely-sound semantics in the presence of NKCIDs with respect to $\mathcal{K}$.

THEOREM 5.3. *Let $\mathcal{DB} = \langle \mathcal{S}, \mathcal{I}, \mathcal{K} \rangle$ be a NKC database schema, $\mathcal{D}$ be a database instance for $\mathcal{DB}$, q be a query of arity n over $\mathcal{DB}$, and $\bar{t}$ be a n-tuple of values of Γ. The problem of establishing whether $\bar{t} \in ans_{ls}(q, \mathcal{DB}, \mathcal{D})$ is coNP-complete with respect to data complexity.*

PROOF. Membership in coNP follows from the algorithm $\mathsf{Answer_{LS}}(\mathcal{DB}, \mathcal{D}, q, \bar{t})$ and from Theorem 3.9. Indeed, in the algorithm the problem of establishing whether $\bar{t} \notin ans_{ls}(q, \mathcal{DB}, \mathcal{D})$, that is the complement of our problem, is carried out by guessing a database and checking conditions (1), (2), and (3) that can be verified in polynomial time.

We prove coNP-hardness of the problem even if we restrict to database schemata without IDs. Actually, this hardness result can be immediately derived from the results reported in [10] (although obtained under a different semantics): however, in the following we provide an alternative proof, in which we use a reduction of the 3-colorability problem to our problem. Consider a graph $G = (V, E)$ with a set of vertices V and edges E. We define a database schema $\mathcal{DB} = \langle \mathcal{S}, \emptyset, \mathcal{K} \rangle$ where $\mathcal{S}$ consists of the two binary relations *edge* and *col*, and $\mathcal{K}$ contains the dependency $key(col) = \{1\}$. The instance $\mathcal{D}$ is defined as follows:

$$\begin{array}{rcl} \mathcal{D} & = & \{col(c, i) | i \in \{1, 2, 3\} \text{ and } c \in V\} \cup \\ & & \{edge(x, y) | \langle x, y \rangle \in E\} \end{array}$$

Finally, we define the query

$$q \leftarrow edge(X, Y), col(X, Z), col(Y, Z)$$

We prove that G is 3-colorable (i.e., for each pair of adjacent vertices, the vertices are associated with different colors) if and only if $\langle\rangle \notin ans_{ls}(q, \mathcal{DB}, \mathcal{D})$ (i.e., the boolean query q has an affirmative answer). In fact, it is immediate to verify that, for each possible coloring C of the graph (i.e., a set of pairs of vertices and colors, where the three colors are represented by the values 1,2,3) there exists $\mathcal{B} \in sem_{ls}(\mathcal{DB}, \mathcal{D})$ that exactly corresponds to C, i.e., $col^{\mathcal{B}}$ is exactly the set of pairs in the coloring C. Therefore, if there exists a coloring that is a 3-coloring, then $\langle\rangle \notin q^{\mathcal{B}}$ for some $\mathcal{B} \in sem_{ls}(\mathcal{DB},\mathcal{D})$, consequently $\langle\rangle \notin ans_{ls}(q,\mathcal{DB},\mathcal{D})$. Conversely, it is immediate to verify that, for each $\mathcal{B} \in sem_{ls}(\mathcal{DB},\mathcal{D})$, $col^{\mathcal{B}\cap\mathcal{D}}$ corresponds to a possible coloring of the graph. Hence, if each possible coloring is not a 3-coloring, then $\langle\rangle \in q^{\mathcal{B}}$, therefore $\langle\rangle \in ans_{ls}(q,\mathcal{DB},\mathcal{D})$. □

THEOREM 5.4. *Let $\mathcal{DB} = \langle\mathcal{S},\mathcal{I},\mathcal{K}\rangle$ be a NKC database schema, $\mathcal{D}$ be a database instance for $\mathcal{DB}$, q be a query of arity n over $\mathcal{DB}$, and $\bar{t}$ be a n-tuple of values of Γ. The problem of establishing whether $\bar{t} \in ans_{ls}(q,\mathcal{DB},\mathcal{D})$ is PSPACE-complete with respect to combined complexity.*

PROOF. Hardness follows from Corollary 3.10 and from the fact that, when $\mathcal{D}$ is consistent with $\mathcal{K}$, $\bar{t} \in ans_{ls}(q,\mathcal{DB},\mathcal{D})$ if and only if $\bar{t} \in ans_{s}(q,\mathcal{DB},\mathcal{D})$.

Membership in PSPACE follows from algorithm $\mathsf{Answer_{LS}}(\mathcal{DB},\mathcal{D},q,t)$ and Theorem 3.9. Indeed, it is easy to see that conditions (1), (2), and (3) can be verified in polynomial space, and furthermore NPSPACE=PSPACE [22]. □

5.2 Query answering under the loosely-exact semantics

We now study the query answering problem under the loosely-exact semantics. We recall that, differently from the loosely-sound semantics, in this case IDs can be satisfied by either adding or deleting facts. Hence, $sem_{le}(\mathcal{DB},\mathcal{D})$ accounts for databases that minimize both elimination and insertion of facts, i.e., that are "as exact as possible".

We first prove that query answering under the loosely-exact semantics is undecidable in the general case, i.e., when no restriction is imposed on the form of IDs and KDs.

THEOREM 5.5. *Let $\mathcal{DB} = \langle\mathcal{S},\mathcal{I},\mathcal{K}\rangle$ be a database schema, $\mathcal{D}$ a database instance for $\mathcal{DB}$, q a query of arity n over $\mathcal{DB}$, and $\bar{t}$ a n-tuple of values of Γ. The problem of establishing whether $\bar{t} \in ans_{le}(q,\mathcal{DB},\mathcal{D})$ is undecidable.*

PROOF. We reduce query answering in the loosely-sound semantics to query answering in the loosely-exact semantics. We can restrict to instances $\mathcal{D}$ consistent with $\mathcal{K}$, since by Theorem 5.1 for this class of instances the problem of establishing whether $\bar{t} \in ans_{ls}(q,\mathcal{DB},\mathcal{D})$ is undecidable. Starting from such a problem instance $(\mathcal{DB},\mathcal{D},q,\bar{t})$, we define a new problem instance $(\mathcal{DB}',\mathcal{D}',q',\bar{t}')$ such that $\bar{t} \in ans_{ls}(q,\mathcal{DB},\mathcal{D})$ iff $\bar{t}' \in ans_{le}(q',\mathcal{DB}',\mathcal{D}')$. Precisely:

- $\mathcal{DB}' = \langle\mathcal{S}',\mathcal{I}',\mathcal{K}'\rangle$ is obtained from $\mathcal{DB}$ by:
  - defining $\mathcal{S}'$ as the schema obtained from $\mathcal{S}$ by adding an attribute to each relation in $\mathcal{S}$ (in the last position);
  - changing each inclusion in order to propagate such a new attribute from r to s, i.e., $\mathcal{I}'$ is obtained from $\mathcal{I}$ by replacing each $I = r[i_1,\ldots,i_k] \subseteq s[j_1,\ldots,j_k]$ with $I' = r[i_1,\ldots,i_k,n] \subseteq s[j_1,\ldots,j_k,m]$, where n is the arity of r in $\mathcal{S}'$ and m is the arity of s in $\mathcal{S}'$;
- $\mathcal{D}'$ is the set $\mathcal{D}'_1 \cup \mathcal{D}'_2$, where $\mathcal{D}'_1 = \{\ r(\overline{u},t_0) | r(\overline{u}) \in \mathcal{D}\ \}$ and
$$\mathcal{D}'_2 = \{\ r(\overline{u},t_1) \mid r \in \mathcal{S} \text{ and } \overline{u} \text{ is a tuple of values of } \Gamma_{\mathcal{D}} \cup \{t_1\}\ \}$$
where $\Gamma_{\mathcal{D}}$ denotes the set of symbols from Γ appearing in $\mathcal{D}$, and t_0, t_1 are values not belonging to $\Gamma_{\mathcal{D}}$. Notice that the set $\mathcal{D}'$ is finite;
- if the query q has the form
$$q(\vec{\mathbf{x}}) \ \leftarrow\ conj_1(\vec{\mathbf{x}},\vec{\mathbf{y}}_1) \vee \cdots \vee conj_k(\vec{\mathbf{x}},\vec{\mathbf{y}}_k)$$
the query q' is as follows:
$$q'(\vec{\mathbf{x}},Y) \leftarrow conj_1(\vec{\mathbf{x}},\vec{\mathbf{y}}_1,t_0) \vee \cdots \vee conj_k(\vec{\mathbf{x}},\vec{\mathbf{y}}_k,t_0) \vee body'$$
where $body'$ is the disjunction
$$\bigvee\{r(\overline{u},t_1) \ \mid\ r(\overline{u}) \in \mathcal{D} \text{ and there is a KD for } r \text{ in } \mathcal{K}\}$$
- $\bar{t}'$ is obtained from $\bar{t}$ by adding the value t_0 at the end of the tuple $\bar{t}$.

It can be shown that $\bar{t} \in ans_{ls}(q,\mathcal{DB},\mathcal{D})$ iff $\bar{t}' \in ans_{le}(q',\mathcal{DB}',\mathcal{D}')$, since for each database $\mathcal{B}$ in $sem_{le}(\mathcal{DB},\mathcal{D})$, there are two possible cases:

1. $\mathcal{B}\cap\mathcal{D} = \mathcal{D}$. In this case, due to the key dependencies $\mathcal{K}$, $\mathcal{B}$ does not contain any tuple of the form $r(\overline{u},t_1)$ such that $r(\overline{u}) \in \mathcal{D}$ and a key dependency for r is defined in $\mathcal{K}$. Consequently, $\bar{t}' \in q'^{\mathcal{B}}$ iff $\bar{t} \in q^{\mathcal{B}}$. Moreover, it is immediate to verify that there exists at least one such $\mathcal{B}$ in $sem_{le}(\mathcal{DB}',\mathcal{D}')$;
2. $\mathcal{B}\cap\mathcal{D} \subset \mathcal{D}$. In this case, there exists at least one tuple in $\mathcal{B}$ of the form $r(\overline{u},t_1)$ such that $r(\overline{u}) \in \mathcal{D}$ and a key dependency for r is defined in $\mathcal{K}$, consequently $\bar{t}' \in q'^{\mathcal{B}}$ for each such $\mathcal{B}$. In other words, this kind of databases does not affect $ans_{le}(q',\mathcal{DB}',\mathcal{D}')$, since in $\mathcal{B}$ every possible tuple is in the answer of q'.

Therefore, $\bar{t} \in ans_{ls}(q,\mathcal{DB},\mathcal{D})$ iff $\bar{t}' \in ans_{le}(q',\mathcal{DB}',\mathcal{D}')$.

Finally, since the above reduction is effectively computable and since, by Theorem 5.1, establishing whether $\bar{t} \in ans_{ls}(q,\mathcal{DB},\mathcal{D})$ is undecidable, the thesis follows. □

Differently from the previous semantics, in the case when the instance $\mathcal{D}$ is consistent with $\mathcal{K}$, we obtain a surprising result: query answering is decidable under the loosely-exact semantics even without any restriction on the form of KDs and IDs.

THEOREM 5.6. *Let $\mathcal{DB} = \langle\mathcal{S},\mathcal{I},\mathcal{K}\rangle$ be a database schema, $\mathcal{D}$ a database instance consistent with $\mathcal{K}$, q a query of arity n over $\mathcal{DB}$, and $\bar{t}$ be a n-tuple of values of Γ. The problem of establishing whether $\bar{t} \in ans_{le}(q,\mathcal{DB},\mathcal{D})$ can be decided in polynomial time with respect to data complexity and is NP-complete with respect to combined complexity.*

PROOF. To prove the thesis, we define the following algorithm:

```
Algorithm AnswerCons_LE($\mathcal{DB},\mathcal{D},q,\bar{t}$)
Input: database schema $\mathcal{DB} = \langle\mathcal{S},\mathcal{I},\mathcal{K}\rangle$,
       instance $\mathcal{D}$ consistent with $\mathcal{K}$,
       conjunctive query $q$ of arity $n$, $n$-tuple $\bar{t}$
Output: true if $\bar{t} \in ans_{le}(q,\mathcal{DB},\mathcal{D})$, false otherwise
$\mathcal{D}_1 = \mathcal{D}$;
repeat
  $\mathcal{D}_0 = \mathcal{D}_1$;
  for each $r(\bar{t}') \in \mathcal{D}_1$
```

```
        if there exists $r[i_1,\ldots,i_k] \subseteq s[j_1,\ldots,j_k] \in \mathcal{I}$
            such that
                for each $s(\bar{t}'') \in \mathcal{D}_1$, $\bar{t}''[j_1,\ldots,j_k] \neq \bar{t}'[i_1,\ldots,i_k]$
        then $\mathcal{D}_1 = \mathcal{D}_1 - \{r(\bar{t}')\}$
until $\mathcal{D}_1 = \mathcal{D}_0$;
if $\bar{t} \in q^{\mathcal{D}_1}$
then return true
else return false
```

Correctness of the algorithm $\mathsf{AnswerCons}_{\mathsf{LE}}$ follows from the fact that the database $\mathcal{D}_1$ computed by the algorithm is such that *(i)* $\mathcal{D}_1 \in sem_{le}(\mathcal{DB},\mathcal{D})$; *(ii)* for each $\mathcal{B} \in sem_{le}(\mathcal{DB},\mathcal{D})$, $\mathcal{B} \supseteq \mathcal{D}_1$. Therefore, $\bar{t} \in ans_{le}(q,\mathcal{DB},\mathcal{D})$ if and only if $\bar{t} \in q^{\mathcal{D}_1}$. It is well-known that this last condition (corresponding to standard query answering over a relational database) can be computed in polynomial time with respect to data complexity and in nondeterministic polynomial time with respect to combined complexity. □

Let us turn our attention on query answering under the loosely-exact semantics in the case of NKC database schemata. To this aim, we first define a particular query $Q(I,\bar{t})$ associated with a tuple $\bar{t}$ and an inclusion dependency I.

Definition 5.7. *Let I be an inclusion dependency of the form $r[i_1,\ldots,i_k] \subseteq s[j_1,\ldots,j_k]$, where r has arity n and s has arity m, and let $\bar{t}$ be an n-tuple. We denote as $Q(I,\bar{t})$ the boolean conjunctive query $q \leftarrow s(z_1,\ldots,z_m)$, where, for each ℓ such that $1 \leq \ell \leq m$, each z_ℓ is as follows: if there exists h such that $\ell = j_h$ then $z_\ell = t[i_h]$, otherwise $z_\ell = X_\ell$.*

In the following, the query $Q(\mathcal{I},\bar{t})$ is used in order to verify whether a database schema $\mathcal{DB}$ and an instance $\mathcal{D}$ imply the existence in all databases $\mathcal{B} \in sem_s(\mathcal{DB},\mathcal{D})$ of a fact of the form $s(\bar{t}')$ such that $\bar{t}[i_1,\ldots,i_k] = \bar{t}'[j_1,\ldots,j_k]$.

Below we define the algorithm $\mathsf{Answer}_{\mathsf{LE}}$ for query answering under the loosely-exact semantics.

```
Algorithm $\mathsf{Answer}_{\mathsf{LE}}(\mathcal{DB},\mathcal{D},q,\bar{t})$
Input: NKC database schema $\mathcal{DB} = \langle\mathcal{S},\mathcal{I},\mathcal{K}\rangle$, instance $\mathcal{D}$,
       query $q$ of arity $n$ over $\mathcal{DB}$, $n$-tuple $\bar{t}$ of values of $\Gamma$
Output: true if $\bar{t} \in ans_{le}(q,\mathcal{DB},\mathcal{D})$, false otherwise
if there exists $\mathcal{D}' \subseteq \mathcal{D}$ such that
   (a) $\mathcal{D}'$ is consistent with $\mathcal{K}$ and
   (b) $\mathsf{Answer}_{\mathsf{S}}(\langle\mathcal{S},\mathcal{I},\emptyset\rangle,\mathcal{D}',q,\bar{t})$ returns false and
   (c) for each $\mathcal{D}''$ such that $\mathcal{D}' \subset \mathcal{D}'' \subseteq \mathcal{D}$
       (c1) $\mathcal{D}''$ is not consistent with $\mathcal{K}$ or
       (c2) there exists $I \in \mathcal{I}$ and $r(\bar{t}_1) \in \mathcal{D}''$
            such that
               $\mathsf{Answer}_{\mathsf{S}}(\langle\mathcal{S},\mathcal{I},\emptyset\rangle,\mathcal{D}',Q(I,\bar{t}_1),\langle\rangle)$ returns false and
               $\mathsf{Answer}_{\mathsf{S}}(\langle\mathcal{S},\emptyset,\emptyset\rangle,\mathcal{D}'',Q(I,\bar{t}_1),\langle\rangle)$ returns false
then return false
else return true
```

Intuitively, to return *false* the algorithm looks for the existence of a database $\mathcal{B}'$ in $sem_{le}(\mathcal{DB},\mathcal{D})$ such that $\bar{t} \notin q^{\mathcal{B}'}$. As in the algorithm $\mathsf{Answer}_{\mathsf{LS}}$, the database $\mathcal{B}'$ is represented by its intersection with the initial instance $\mathcal{D}$ (denoted as $\mathcal{D}'$ in the algorithm): the fact that $\bar{t} \notin q^{\mathcal{B}'}$ is verified by condition (b), while the fact that $\mathcal{B}' \in sem_{le}(\mathcal{DB},\mathcal{D})$ is verified by conditions (a) and (c) of the algorithm. In particular, condition (c) verifies that, for each database $\mathcal{B}''$ (represented by its intersection with $\mathcal{D}$ denoted as $\mathcal{D}''$), it is not the case that $\mathcal{B}'' \gg_{\mathcal{D}} \mathcal{B}'$. In conditions (c1) and (c2), the symbol $\langle\rangle$ denotes the empty tuple.

Soundness and completeness of the algorithm is established by the following theorem.

Theorem 5.8. *Let $\mathcal{DB} = \langle\mathcal{S},\mathcal{I},\mathcal{K}\rangle$ be a NKC database schema, $\mathcal{D}$ be a database instance, q be a query of arity n over $\mathcal{S}$, and $\bar{t}$ be a n-tuple of values from Γ. Then, $\bar{t} \in ans_{le}(q,\mathcal{DB},\mathcal{D})$ iff $\mathsf{Answer}_{\mathsf{LE}}(\mathcal{DB},\mathcal{D},q,\bar{t})$ returns true.*

Proof. In order to prove correctness of the above algorithm, we need a preliminary lemma. In the following, given an instance $\mathcal{D}$ of a database schema $\mathcal{DB} = \langle\mathcal{S},\mathcal{I},\emptyset\rangle$, we denote as $chase_1(\mathcal{DB},\mathcal{D})$ the set of new facts obtained by applying the chase rule to the facts in $\mathcal{D}$, i.e., the set of facts of the form $s(\bar{t}_2)$ such that there exist $I = r[i_1,\ldots,i_k] \subseteq s[j_1,\ldots,j_k] \in \mathcal{I}$ and $r(\bar{t}_1) \in \mathcal{D}$ such that $\bar{t}_1[i_1,\ldots,i_k] = \bar{t}_2[j_1,\ldots,j_k]$ and there exists no $s(\bar{t}_3) \in \mathcal{D}$ such that $\bar{t}_1[i_1,\ldots,i_k] = \bar{t}_3[j_1,\ldots,j_k]$.

Lemma 5.9. *Let $\mathcal{D}',\mathcal{D}''$ be instances of a database schema $\mathcal{DB} = \langle\mathcal{S},\mathcal{I},\emptyset\rangle$ such that $\mathcal{D}' \subset \mathcal{D}''$ and, for each $I \in \mathcal{I}$ of the form $I = r[i_1,\ldots,i_k] \subseteq s[j_1,\ldots,j_k]$ and for each $r(\bar{t}_1) \in \mathcal{D}''$, either $\mathsf{Answer}_{\mathsf{S}}(\langle\mathcal{S},\mathcal{I},\emptyset\rangle,\mathcal{D}',Q(I,\bar{t}_1),\langle\rangle)$ returns true or $\mathsf{Answer}_{\mathsf{S}}(\langle\mathcal{S},\emptyset,\emptyset\rangle,\mathcal{D}'',Q(I,\bar{t}_1),\langle\rangle)$ returns true. Then, $chase(\mathcal{DB},\mathcal{D}'') - \mathcal{D}'' \subseteq chase(\mathcal{DB},\mathcal{D}') - \mathcal{D}'$.*

Proof. It is straightforward to verify that the hypothesis implies that $chase_1(\mathcal{DB},\mathcal{D}'') \subseteq chase_1(\mathcal{DB},\mathcal{D}')$; this in turn implies that each new fact added in $chase(\mathcal{DB},\mathcal{D}'')$ by an application of the chase rule in $chase(\mathcal{DB},\mathcal{D}'')$ is also added by the chase rule in $chase(\mathcal{DB},\mathcal{D}')$. Consequently, the thesis follows. □

We now prove the theorem.

"⇒" Suppose $\mathsf{Answer}_{\mathsf{LE}}(\mathcal{DB},\mathcal{D},q,\bar{t})$ returns *false*. Then, there exists $\mathcal{D}' \subseteq \mathcal{D}$ such that conditions (a), (b) and (c) of the algorithm hold for $\mathcal{D}'$. Let $B' = chase(\mathcal{DB},\mathcal{D}')$. Now, suppose $\mathcal{B}' \notin sem_{le}(\mathcal{DB},\mathcal{D})$: hence, there exists a database instance $\mathcal{B}''$ such that $\mathcal{B}''$ is consistent with $\mathcal{K}$ and $\mathcal{B}'' \gg_{\mathcal{D}} \mathcal{B}'$, which implies that $\mathcal{B}'' - \mathcal{D} \subseteq \mathcal{B}' - \mathcal{D}$. Since by hypothesis condition (c) holds for $\mathcal{D}'$, it follows that condition (c2) holds for $\mathcal{D}''$, i.e., there exists a fact $r(\bar{t}_1) \in \mathcal{D}''$ and an inclusion $I = r[i_1,\ldots,i_k] \subseteq s[j_1,\ldots,j_k] \in \mathcal{I}$ such that:

1. $\mathsf{Answer}_{\mathsf{S}}(\langle\mathcal{S},\mathcal{I},\emptyset\rangle,\mathcal{D}',Q(I,\bar{t}_1),\langle\rangle)$ returns *false*, which implies that there is no fact in $\mathcal{B}'$ of the form $s(\bar{t}_2)$ such that $\bar{t}_1[i_1,\ldots,i_k] = \bar{t}_2[j_1,\ldots,j_k]$;

2. $\mathsf{Answer}_{\mathsf{S}}(\langle\mathcal{S},\emptyset,\emptyset\rangle,\mathcal{D}'',Q(I,\bar{t}_1),\langle\rangle)$ returns *false*, which implies that there is no fact in $\mathcal{D}''$ of the form $s(\bar{t}_2)$ such that $\bar{t}_1[i_1,\ldots,i_k] = \bar{t}_2[j_1,\ldots,j_k]$. On the other hand, a fact of the form $s(\bar{t}_2)$ such that $\bar{t}_1[i_1,\ldots,i_k] = \bar{t}_2[j_1,\ldots,j_k]$ must be present in $\mathcal{B}''$, due to the presence of $r(\bar{t}_1)$ in $\mathcal{D}''$ and to the inclusion I.

The two above conditions imply that there exists a fact of the form $s(\bar{t}_2)$ in $\mathcal{B}'' - \mathcal{D}$ which does not belong to $\mathcal{B}'$. Consequently, $\mathcal{B}'' - \mathcal{D} \subseteq \mathcal{B}' - \mathcal{D}$ does not hold, thus contradicting the hypothesis that $\mathcal{B}'' \gg_{\mathcal{D}} \mathcal{B}'$. Therefore, $\mathcal{B}' \in sem_{le}(\mathcal{DB},\mathcal{D})$, and since conditions (a) and (b) hold for $\mathcal{D}'$, it follows that $\bar{t} \notin q^{\mathcal{B}'}$, hence $\bar{t} \notin ans_{le}(q,\mathcal{DB},\mathcal{D})$.

"⇐" Suppose $\bar{t} \notin ans_{le}(q,\mathcal{DB},\mathcal{D})$. Therefore, there exists $\mathcal{B}' \in sem_{le}(\mathcal{DB},\mathcal{D})$ such that $\bar{t} \notin q^{\mathcal{B}'}$. Let $\mathcal{D}' = \mathcal{D} \cap \mathcal{B}'$. Since $\mathcal{B}' \in sem_{le}(\mathcal{DB},\mathcal{D})$, condition (a) of the algorithm holds for $\mathcal{B}'$, and since $\mathcal{D}' \subseteq \mathcal{B}'$, condition (a) holds for $\mathcal{D}'$ as well. From $\bar{t} \notin q^{\mathcal{B}'}$ and from soundness and completeness of the algorithm $\mathsf{Answer}_{\mathsf{S}}$ it follows that condition (b) holds for $\mathcal{D}'$. Now, suppose condition (c) does not hold for $\mathcal{D}'$: then, there exists $\mathcal{D}''$ such that conditions (c1) and (c2) do not hold for $\mathcal{D}'$ and $\mathcal{D}''$, i.e., $\mathcal{D}''$ is consistent with $\mathcal{K}$ and, for each $I \in \mathcal{I}$ of the form $I = r[i_1, \ldots, i_k] \subseteq s[j_1, \ldots, j_k]$ and for each $r(\overline{t}_1) \in \mathcal{D}''$, either $\mathsf{Answer_S}(\langle \mathcal{S}, \mathcal{I}, \emptyset\rangle, \mathcal{D}', Q(I, \overline{t}_1), \langle\rangle)$ returns *true* or $\mathsf{Answer_S}(\langle \mathcal{S}, \emptyset, \emptyset\rangle, \mathcal{D}'', Q(I, \overline{t}_1), \langle\rangle)$ returns *true*. By Lemma 5.9, it follows that $chase(\mathcal{DB}, \mathcal{D}'') - \mathcal{D}'' \subseteq chase(\mathcal{DB}, \mathcal{D}') - \mathcal{D}'$. Now let $\mathcal{B}'' = chase(\mathcal{DB}, \mathcal{D}'')$: since $\mathcal{B}' \supseteq chase(\mathcal{DB}, \mathcal{D}')$, it follows that $\mathcal{B}'' - \mathcal{D} \subseteq \mathcal{B}' - \mathcal{D}$, and by hypothesis $\mathcal{D}'' \supset \mathcal{D}'$, therefore $\mathcal{B}'' \cap \mathcal{D} \supset \mathcal{B}' \cap \mathcal{D}$, hence $\mathcal{B}'' \gg_{\mathcal{D}} \mathcal{B}'$. Moreover, since $\mathcal{D}''$ is consistent with $\mathcal{K}$, $\mathcal{B}''$ is consistent with $\mathcal{K}$ and $\mathcal{I}$, consequently $\mathcal{B}' \notin sem_{le}(\mathcal{DB}, \mathcal{D})$, thus contradicting the hypothesis. Therefore, condition (c) holds for $\mathcal{D}'$, which implies that $\mathsf{Answer_{LE}}(\mathcal{DB}, \mathcal{D}, q, \overline{t})$ returns *false*. $\Box$

Finally, based on the above algorithm, we analyze the computational complexity of query answering under the loosely-exact semantics for NKC database schemata.

THEOREM 5.10. *Let $\mathcal{DB} = \langle \mathcal{S}, \mathcal{I}, \mathcal{K}\rangle$ be a NKC database schema, $\mathcal{D}$ a database instance, q a query of arity n over $\mathcal{DB}$, and $\overline{t}$ a n-tuple of values of Γ. The problem of establishing whether $\overline{t} \in ans_{le}(q, \mathcal{DB}, \mathcal{D})$ is Π_2^p-complete with respect to data complexity and PSPACE-complete with respect to combined complexity.*

Proof sketch. The analysis of the algorithm $\mathsf{Answer_{LE}}$ shows that the problem is in Π_2^p with respect to data complexity. Indeed, it is immediate to verify that:

- condition (a) can be verified in polynomial time;
- condition (b) can be verified in polynomial time, as shown in Section 3;
- conditions (c1) and (c2) can be verified in polynomial time: therefore, condition (c) can be verified in nondeterministic polynomial time.

Consequently, if considered as a nondeterministic procedure, the algorithm runs in Π_2^p with respect to data complexity. Hardness with respect to Π_2^p can be proved by a reduction from 2-QBF validity, i.e., the validity problem for quantified boolean formulae having the form $\forall \overline{x} \exists \overline{y} f(\overline{x}, \overline{y})$ where $f(\overline{x}, \overline{y})$ is a 3-CNF, i.e., a propositional formula in 3-conjunctive normal form. The reduction generalizes the scheme employed in the proof of Theorem 5.3.

As concerns combined complexity, it is immediate to verify that each of the conditions of the algorithm is computed in nondeterministic polynomial space, therefore the algorithm runs in nondeterministic polynomial space with respect to combined complexity, which proves membership in PSPACE of the problem. PSPACE-hardness can be proved by reducing query answering under loosely-sound semantics for databases without key dependencies to this problem. The reduction is obtained by a slight modification of the reduction from query answering under loosely-sound semantics exhibited in the proof of Theorem 5.5, and observing that, if the original problem instance is such that, for each $I = r[\vec{\mathbf{A}}] \subseteq s[\vec{\mathbf{B}}] \in \mathcal{I}$, $\vec{\mathbf{B}}$ does not cover the set of all the attributes of s, then the derived database schema $\mathcal{DB}'$ is a NKC schema. Moreover, it is immediate to verify that restricting to such a kind of problem instances does not affect PSPACE-hardness of the query answering problem under the loosely-sound semantics. Finally, the reduction is modified in a way such that the database instance $\mathcal{D}'$ obtained from the original instance $\mathcal{D}$ has size polynomial with respect to data complexity.

6. DISCUSSION

6.1 Summary of results

The summary of the results we have obtained is reported in Figure 1[3], in which we have two distinct tables, that present, respectively, the complexity of query answering for the class of general database instances and for instances consistent with KDs. Each column (with the exception of the first two) corresponds to a different semantics, while each row corresponds to a different class of dependencies (specified in the first two columns). Each cell of the tables reports data complexity and combined complexity of query answering: for each decidable case, the complexity of the problem is complete with respect to the class reported. We have marked with the symbol ♠ the cells corresponding either to already known results or to results straightforwardly implied by known results.

We point out that, due to the correspondence between query answering and query containment illustrated in Section 3, all the complexity results established for the problem of query answering also hold for the conjunctive query containment problem.

6.2 Related work

The problem of reasoning with inconsistent databases is closely related to the studies in *belief revision and update* [2]. This area of Artificial Intelligence studies the problem of integrating new information with previous knowledge. In general, the problem is studied in a logical framework, in which the new information is a logical formula f and the previous knowledge is a logical theory (also called knowledge base) T. Of course, f may in general be inconsistent with T. The *revised* (or *updated*) knowledge base is denoted as $T \circ f$, and several semantics have been proposed for the operator $\circ$. The semantics for belief revision can be divided into *revision* semantics, when the new information f is interpreted as a modification of the knowledge about the world, and *update* semantics, when f reflects a change in the world.

The problem of reasoning with inconsistent databases can be actually seen as a problem of belief revision. In fact, with respect to the above illustrated knowledge base revision framework, if we consider the database instance $\mathcal{D}$ as the initial knowledge base T, and the set of integrity constraints $\mathcal{I} \cup \mathcal{K}$ as the new information f, then the problem of deciding whether a tuple t is in the answer set of a query q with respect to the database schema $\mathcal{DB} = \langle \mathcal{S}, \mathcal{I}, \mathcal{K}\rangle$ and the instance $\mathcal{D}$ corresponds to the belief revision problem $\mathcal{D} \circ (\mathcal{I} \cup \mathcal{K}) \models q(t)$. Based on such a correspondence, the studies in belief revision appear very relevant for the field of inconsistent databases: indeed, almost all the approaches to inconsistent databases that we have considered in this section can be reconstructed in terms of direct applications of well-known semantics for belief revision/update in a particular class of theories.

On the other hand, from a computational perspective, there are no results concerning the particular kind of belief revision/update that is of interest for database applications: in particular, the class of relational integrity constraints as revision/update knowledge has not been taken into account in the belief revision literature, where the computational

[3]Due to space limitations, in the present version of the paper we have not been able to include the proofs of all the results reported in Figure 1.

Data complexity/combined complexity for *general* database instances:

KDs	IDs	strictly-sound	loosely-sound	loosely-exact
no	GEN	PTIME/PSPACE♠	PTIME/PSPACE	PTIME/NP
yes	no	PTIME/NP♠	coNP/Π_2^p♠	coNP/Π_2^p♠
yes	FK	PTIME/PSPACE	coNP/PSPACE	coNP/PSPACE
yes	FK,UN	PTIME/PSPACE	coNP/PSPACE	Π_2^p/PSPACE
yes	NKC	PTIME/PSPACE	coNP/PSPACE	Π_2^p/PSPACE
yes	1KC	undecidable	undecidable	undecidable
yes	GEN	undecidable♠	undecidable	undecidable

Data complexity/combined complexity for *key-consistent* database instances:

KDs	IDs	strictly-sound	loosely-sound	loosely-exact
no	GEN	PTIME/PSPACE♠	PTIME/PSPACE	PTIME/NP
yes	no	PTIME/NP♠	PTIME/NP♠	PTIME/NP♠
yes	FK	PTIME/PSPACE	PTIME/PSPACE	PTIME/NP
yes	FK,UN	PTIME/PSPACE	PTIME/PSPACE	PTIME/NP
yes	NKCID	PTIME/PSPACE	PTIME/PSPACE	PTIME/NP
yes	1KCID	undecidable	undecidable	PTIME/NP
yes	GEN	undecidable♠	undecidable	PTIME/NP

Legenda: FK = foreign key dependencies, GEN = general IDs, UN = unary IDs; ♠ = already known result.

Figure 1: Complexity of query answering under KDs and IDs (decision problem)

results mostly concern a setting in which knowledge is specified in terms of propositional formulae of classical logic [12, 13]. Instead, the typical database setting is considered by the literature on *inconsistent databases*, which we briefly survey in the following.

The notion of consistent query answers over inconsistent databases was originally given in [5]. However, the approach in [5] is completely proof-theoretic, and no computational technique for obtaining consistent answers from inconsistent database is provided.

In [21] the authors describe an operator for *merging databases* under constraints which allows for obtaining a maximal amount of information from each database by means of a majority criterion used in case of conflict. Even if a large set of constraints is considered, namely the constraints that can be expressed as first-order formulae, the computational complexity of the merging procedure is not explored, and no algorithm to compute consistent query answers is provided. Furthermore, the problem of dealing with incomplete databases is not taken into account. Notice also that, different from all the other studies mentioned in the following, this approach relies on a cardinality-based ordering between databases (rather than a set-containment-based ordering).

In [15] the authors propose a framework for updating theories and logical databases (i.e., databases obtained by giving priorities to sentences in the databases) that can be extended also to the case of updating views. The semantics proposed in such a paper is based on a particular set-containment based ordering between theories that "accomplish" an update to an original theory, which is similar to the loosely-sound semantics above presented.

In [3] the authors define an algorithm for consistent query answers in inconsistent databases based on the notion of *residues*, originally defined in the context of semantic query optimization. The method is proved to be sound and complete only for the class of universally quantified binary constraints, i.e., constraints that involve two database relations. In [4] the same authors propose a new method that can handle arbitrary universally quantified constraints by specifying the database repairs into *logic rules with exceptions* (LPe). The semantics underlying the notion of consistent query answers both in [3] and in [4] is defined on a set-containment ordering between databases, which corresponds to the loosely-exact semantics of our framework.

Moreover, a different semantics for database repairing has been considered in [10, 9]. Specifically, in such works a semantics is defined in which only tuple elimination is allowed; therefore, the problem of dealing with infinite models is not addressed. Then, a preference order over the database repairs is defined, in such a way that only minimal repairs (in terms of set containment) are considered. Hence, the semantics is a "maximal complete" one, in the sense that only maximal consistent subsets of the database instance are considered as repairs of such an instance. In [10] the authors establish complexity results for query answering under such a semantics in the presence of *denial constraints*, a generalization of key dependencies and functional dependencies, while in [9] also inclusion dependencies are considered. Such a "maximal complete" semantics is different from the complete semantics considered in the present paper.

Finally, [16] proposes a technique to deal with inconsistencies that is based on the reformulation of integrity constraints into a disjunctive datalog program with two different forms of negation: negation as failure and classical negation. Such a program can be used both to repair databases, i.e., modify the data in the databases in order to satisfy integrity constraints, and to compute consistent query answers. The technique is proved to be sound and complete for universally quantified constraints. The semantics adopted to support this method corresponds to our loosely-exact semantics.

We point out that none of the above mentioned works provides a general solution for the case of cyclic inclusion dependencies under the semantics (both strict and loose) considered in this paper.

6.3 Future work

Although obtained in a single database context, many of the techniques and results presented here are directly applicable to data integration, where multiple information sources may provide data that are inconsistent with respect to the global view of the sources. Indeed, we believe that one important development of the research presented in this paper is towards both the computational analysis of query answering in data integration systems and the definition of effective query processing techniques in such a setting.

Moreover, we are currently working on the extension of the present framework with more complex forms of dependencies, e.g., functional dependencies and exclusion dependencies.

7. ACKNOWLEDGMENTS

This research has been supported by the Projects INFOMIX (IST-2001-33570) and SEWASIE (IST-2001-34825) funded by the EU, and by the Project D2I funded by MIUR (Ministero per l'Istruzione, l'Università e la Ricerca). We would like to thank Maurizio Lenzerini and Giuseppe De Giacomo for their insightful comments about this material.

8. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison Wesley Publ. Co., Reading, Massachussetts, 1995.

[2] C. E. Alchourrón, P. Gärdenfors, and D. Makinson. On the logic of theory change: Partial meet contraction and revision functions. *J. of Symbolic Logic*, 50:510–530, 1985.

[3] M. Arenas, L. E. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *Proc. of the 18th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'99)*, pages 68–79, 1999.

[4] M. Arenas, L. E. Bertossi, and J. Chomicki. Specifying and querying database repairs using logic programs with exceptions. In *Proc. of the 4th Int. Conf. on Flexible Query Answering Systems (FQAS 2000)*, pages 27–41. Springer, 2000.

[5] F. Bry. Query answering in information systems with integrity constraints. In *IFIP WG 11.5 Working Conf. on Integrity and Control in Information System*. Chapman & Hall, 1997.

[6] A. Calì, D. Calvanese, G. De Giacomo, and M. Lenzerini. Accessing data integration systems through conceptual schemas. In *Proc. of the 20th Int. Conf. on Conceptual Modeling (ER 2001)*, pages 270–284, 2001.

[7] A. Calì, D. Calvanese, G. De Giacomo, and M. Lenzerini. Data integration under integrity constraints. *Information Systems*, 2003. To appear.

[8] M. A. Casanova, R. Fagin, and C. H. Papadimitriou. Inclusion dependencies and their interaction with functional dependencies. *J. of Computer and System Sciences*, 28(1):29–59, 1984.

[9] J. Chomicki and J. Marcinkowski. Minimal-change integrity maintenance using tuple deletions. Technical Report `arXiv:cs.DB/0212004v1`, 2002.

[10] J. Chomicki and J. Marcinkowski. On the computational complexity of consistent query answers. Technical Report `arXiv:cs.DB/0204010v1`, 2002.

[11] P. M. Dung. Integrating data from possibly inconsistent databases. In *Proc. of the 4th Int. Conf. on Cooperative Information Systems (CoopIS'96)*, pages 58–65, 1996.

[12] T. Eiter and G. Gottlob. On the complexity of propositional knowledge base revision, updates and counterfactuals. *Artificial Intelligence*, 57:227–270, 1992.

[13] T. Eiter and G. Gottlob. The complexity of nested counterfactuals and iterated knowledge base revisions. *J. of Computer and System Sciences*, 53(3):497–512, 1996.

[14] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: Semantics and query answering. In *Proc. of the 9th Int. Conf. on Database Theory (ICDT 2003)*, pages 207–224, 2003.

[15] R. Fagin, J. D. Ullman, and M. Y. Vardi. On the semantics of updates in databases. In *Proc. of the 2nd ACM SIGACT SIGMOD Symp. on Principles of Database Systems (PODS'83)*, pages 352–365, 1983.

[16] G. Greco, S. Greco, and E. Zumpano. A logic programming approach to the integration, repairing and querying of inconsistent databases. In *Proc. of the 17th Int. Conf. on Logic Programming (ICLP'01)*, volume 2237 of *Lecture Notes in Artificial Intelligence*, pages 348–364. Springer, 2001.

[17] A. Y. Halevy. Answering queries using views: A survey. *Very Large Database J.*, 10(4):270–294, 2001.

[18] D. S. Johnson and A. C. Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. *J. of Computer and System Sciences*, 28(1):167–189, 1984.

[19] D. Lembo, M. Lenzerini, and R. Rosati. Source inconsistency and incompleteness in data integration. In *Proc. of the 9th Int. Workshop on Knowledge Representation meets Databases (KRDB 2002)*. CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/Vol-54/`, 2002.

[20] M. Lenzerini. Data integration: A theoretical perspective. In *Proc. of the 21st ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2002)*, pages 233–246, 2002.

[21] J. Lin and A. O. Mendelzon. Merging databases under constraints. *Int. J. of Cooperative Information Systems*, 7(1):55–76, 1998.

[22] C. H. Papadimitriou. *Computational Complexity*. Addison Wesley Publ. Co., Reading, Massachussetts, 1994.

[23] R. Reiter. On closed world data bases. In H. Gallaire and J. Minker, editors, *Logic and Databases*, pages 119–140. Plenum Publ. Co., New York, 1978.

DL-Lite: Tractable Description Logics for Ontologies

Diego Calvanese[1], **Giuseppe De Giacomo**[2], **Domenico Lembo**[2], **Maurizio Lenzerini**[2], **Riccardo Rosati**[2]

[1] Faculty of Computer Science
Free University of Bolzano/Bozen
Piazza Domenicani 3
I-39100 Bolzano, Italy
calvanese@inf.unibz.it

[2] Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113
I-00198 Roma, Italy
lastname@dis.uniroma1.it

Abstract

We propose a new Description Logic, called *DL-Lite*, specifically tailored to capture basic ontology languages, while keeping low complexity of reasoning. Reasoning here means not only computing subsumption between concepts, and checking satisfiability of the whole knowledge base, but also answering complex queries (in particular, conjunctive queries) over the set of instances maintained in secondary storage. We show that in *DL-Lite* the usual DL reasoning tasks are polynomial in the size of the TBox, and query answering is polynomial in the size of the ABox (i.e., in data complexity). To the best of our knowledge, this is the first result of polynomial data complexity for query answering over DL knowledge bases. A notable feature of our logic is to allow for a separation between TBox and ABox reasoning during query evaluation: the part of the process requiring TBox reasoning is independent of the ABox, and the part of the process requiring access to the ABox can be carried out by an SQL engine, thus taking advantage of the query optimization strategies provided by current DBMSs.

Introduction

One of the most important lines of research in Description Logics (DLs) is concerned with the trade-off between expressive power and computational complexity of sound and complete reasoning. Research carried out in the past on this topic has shown that many DLs with efficient, i.e., worst-case polynomial time, reasoning algorithms lack modeling power required in capturing conceptual models and basic ontology languages, while most DLs with sufficient modeling power suffer from inherently worst-case exponential time behavior of reasoning [4, 5].

Although the requirement of polynomially tractable reasoning might be less stringent when dealing with relatively small ontologies, we believe that the need of efficient reasoning algorithms is of paramount importance when the ontology system is to manage large amount of objects (e.g., from thousands to millions of instances). This is the case of several important applications where the use of ontologies is advocated nowadays. For example, in the Semantic Web, ontologies are often used to describe the relevant concepts of Web repositories, and such repositories may incorporate very large data sets, which constitute the instances of the concepts in the ontology. In such cases, two requirements emerge that are typically overlooked in DLs. First, the number of objects in the knowledge bases requires managing instances of concepts (i.e., ABoxes) in secondary storage. Second, significant queries to be posed to the knowledge bases are more complex than the simple queries (i.e., concepts and roles) usually considered in DL research. Unfortunately, in these contexts, whenever the complexity of reasoning is exponential in the size of the instances (as for example in Fact[1], Racer[2] and in [11]), there is little hope for effective instance management and query answering algorithms.

In this paper we propose a new DL, called *DL-Lite*, specifically tailored to capture basic ontology languages, while keeping low complexity of reasoning, in particular, polynomial in the size of the instances in the knowledge base. Reasoning here means not only computing subsumption between concepts, and checking satisfiability of the whole knowledge base, but also answering complex queries over the set of instances maintained in secondary storage.

Our contributions are the following:

1. We define *DL-Lite*, and show that it is rich enough to capture a significant ontology language. Although at a first sight *DL-Lite* appears to be a very simple DL, the kind of modeling constructs in our logic makes it suitable for expressing a variety of representation languages widely adopted in different contexts, such as basic ontology languages, conceptual data models (e.g., Entity-Relationship [2]), and object-oriented formalisms (e.g., basic UML class diagrams[3]).

2. For such a DL we propose novel reasoning techniques for a variety of tasks, including conjunctive query answering and containment between conjunctive queries over concepts and roles. Our presentation is focused especially on the problem of answering conjunctive queries over a knowledge base. We observe that this is one of the few results on answering complex queries (i.e., not corresponding simply to a concept or a role) over a DL knowledge base [11]. Indeed, answering conjunctive queries over a knowledge base is a challenging problem, even in the case

[1] http://www.cs.man.ac.uk/~horrocks/FaCT/

[2] http://www.sts.tu-harburg.de/~r.f.moeller/racer/

[3] http://www.omg.org/uml/

of *DL-Lite*, where the combination of constructs expressible in the knowledge base does not pose particular difficulties in computing subsumption. Notice that, in spite of the simplicity of *DL-Lite* TBoxes, the ability of taking TBox knowledge into account during the process of answering conjunctive queries goes beyond the "variable-free" fragments of first-order logic represented by DLs.

3. An important feature of our approach is that it is perfectly suited to representing ABox assertions managed in secondary storage by a Data Base Management System (DBMS). Indeed, our query answering algorithm is based on the idea of expanding the original query into a set of queries that can be directly evaluated by an SQL engine over the ABox, thus taking advantage of well established query optimization strategies. Notably, this was one of the motivations behind several research works done on CLASSIC in the 80's [6].

4. We analyze the complexity of reasoning in *DL-Lite*. We show that the usual reasoning tasks considered in DLs (i.e., subsumption and satisfiability) can be done in polynomial time. As for query answering, computing the answers to a conjunctive query is worst-case exponential in the size of the TBox and the query, but is polynomial in the size of the ABox, i.e., in data complexity [17]. Hence, the complexity of answering queries is no worse than traditional query evaluation in relational databases[4].

A prototype implementation of *DL-Lite* has been developed and tested within a research project carried out jointly by our institution and the IBM Tivoli Laboratory. First experiments show that our approach is extremely effective: complex domains can be modeled in *DL-Lite*, and it takes no more than a few minutes to answer conjunctive queries over knowledge bases with millions of instances.

DL-Lite

As usual in DLs, *DL-Lite* allows for representing the domain of interest in terms of concepts, denoting sets of objects, and roles, denoting binary relations between objects. *DL-Lite* concepts are defined as follows:

$$\begin{array}{rcl} B & ::= & A \mid \exists R \mid \exists R^- \\ C & ::= & B \mid \neg B \mid C_1 \sqcap C_2 \end{array}$$

where A denotes an atomic concept and R denotes an (atomic) role; B denotes a *basic concept* that can be either an atomic concept, a concept of the form $\exists R$, i.e., the standard DL construct of unqualified existential quantification on roles, or a concept of the form $\exists R^-$, which involves an *inverse role*. C (possibly with subscript) denotes a (general) concept. Note that we use negation of basic concepts only, and we do not allow for disjunction.

A *DL-Lite* knowledge base (KB) is constituted by two components: a TBox used to represent intensional knowledge, and an ABox, used to represent extensional information. *DL-Lite TBox* assertions are of the form

$B \sqsubseteq C$	*inclusion assertions*
$(\mathsf{funct}\ R), (\mathsf{funct}\ R^-)$	*functionality assertions*

[4]We remind the reader that the algorithms for answering a conjunctive query posed to a relational database are exponential in the size of the query.

An inclusion assertion expresses that a basic concept is subsumed by a general concept, while a functionality assertion expresses the (global) functionality of a role, or of the inverse of a role.

As for the ABox, *DL-Lite* allows for assertions of the form:

$$B(a), R(a, b) \quad \textit{membership assertions}$$

where a and b are constants. These assertions state respectively that the object denoted by a is an instance of the basic concept B, and that the pair of objects denoted by (a, b) is an instance of the role R.

Although *DL-Lite* is quite simple from the language point of view, it allows for querying the extensional knowledge of a KB in a much more powerful way than usual DLs, in which only membership to a concept or to a role can be asked. Specifically, *DL-Lite* allows for using conjunctive queries of arbitrary complexity. A conjunctive query (CQ) q over a knowledge base $\mathcal{K}$ is an expression of the form

$$q(\vec{x}) \leftarrow \exists \vec{y}.conj(\vec{x}, \vec{y})$$

where $\vec{x}$ are the so-called *distinguished variables*, $\vec{y}$ are existentially quantified variables called the *non-distinguished* variables, and $conj(\vec{x}, \vec{y})$ is a conjunction of atoms of the form $B(z)$, or $R(z_1, z_2)$, where B and R are respectively a basic concept and a role in $\mathcal{K}$, and z, z_1, z_2 are constants in $\mathcal{K}$ or variables in $\vec{x}$ or $\vec{y}$. Sometimes, for simplifying notation, we will use the Datalog syntax, and write queries of the above form as $q(\vec{x}) \leftarrow body(\vec{x}, \vec{y})$, where the existential quantification $\exists \vec{y}$ has been made implicit, and the symbol "," is used for conjunction in $body(\vec{x}, \vec{y})$.

The semantics of *DL-Lite* is given in terms of interpretations over a fixed infinite *domain* Δ. We assume to have one constant for each object, denoting exactly that object. In other words, we have *standard names* [15], and we will not distinguish between the alphabet of constants and Δ.

An *interpretation* $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}})$ consists of a first order structure over Δ with an *interpretation function* $\cdot^{\mathcal{I}}$ such that:

$$\begin{array}{ll} A^{\mathcal{I}} \subseteq \Delta & R^{\mathcal{I}} \subseteq \Delta \times \Delta \\ (\neg B)^{\mathcal{I}} = \Delta \setminus B^{\mathcal{I}} & (\exists R)^{\mathcal{I}} = \{c \mid \exists c'. (c, c') \in R^{\mathcal{I}}\} \\ (C_1 \sqcap C_2)^{\mathcal{I}} = C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}} & (\exists R^-)^{\mathcal{I}} = \{c \mid \exists c'. (c', c) \in R^{\mathcal{I}}\} \end{array}$$

An interpretation $\mathcal{I}$ is a *model* of an inclusion assertion $B \sqsubseteq C$ iff $B^{\mathcal{I}} \subseteq C^{\mathcal{I}}$; $\mathcal{I}$ is a model of a functionality assertion $(\mathsf{funct}\ R)$ if $(c, c') \in R^{\mathcal{I}} \wedge (c, c'') \in R^{\mathcal{I}} \supset c' = c''$, similarly for $(\mathsf{funct}\ R^-)$; $\mathcal{I}$ is a model of a membership assertion $B(a)$ (resp. $R(a, b)$) if $a \in B^{\mathcal{I}}$ (resp. $(a, b) \in R^{\mathcal{I}}$). A *model of a KB* $\mathcal{K}$ is an interpretation $\mathcal{I}$ that is a model of all the assertions in $\mathcal{K}$. A KB is *satisfiable* if it has at least one model. A KB $\mathcal{K}$ *logically implies* an assertion α if all the models of $\mathcal{K}$ are also models of α. A query $q(\vec{x}) \leftarrow \exists \vec{y}.conj(\vec{x}, \vec{y})$ is interpreted in an interpretation $\mathcal{I}$ as the set $q^{\mathcal{I}}$ of tuples $\vec{c} \in \Delta \times \cdots \times \Delta$ such that when we substitute the variables $\vec{x}$ with the constants $\vec{c}$, the formula $\exists \vec{y}.conj(\vec{x}, \vec{y})$ evaluates to true in $\mathcal{I}$.

Since *DL-Lite* deals with conjunctive queries, the basic reasoning services that are of interest are:

- *query answering*: given a query q with distinguished variables $\vec{x}$ and a KB $\mathcal{K}$, return the set $ans(q, \mathcal{K})$ of tuples $\vec{c}$ of constants of $\mathcal{K}$ such that in every model $\mathcal{I}$ of $\mathcal{K}$ we have $\vec{c} \in q^{\mathcal{I}}$. Note that this task generalizes *instance checking* in DLs, i.e., checking whether a given object is an instance of a specified concept in every model of the knowledge base.
- *query containment*: given two queries q_1 and q_2 and a KB $\mathcal{K}$, verify whether in every model $\mathcal{I}$ of $\mathcal{K}$ $q_1^{\mathcal{I}} \subseteq q_2^{\mathcal{I}}$. Note that this task generalizes *logical implication* of inclusion assertions in DLs.
- *KB satisfiability*: verify whether a KB is satisfiable.

Example 1 Consider the atomic concepts $Professor$ and $Student$, the roles $TeachesTo$ and $HasTutor$, and the following *DL-Lite* TBox $\mathcal{T}$:

$$\begin{array}{ll} Professor \sqsubseteq \exists TeachesTo & Student \sqsubseteq \exists HasTutor \\ \exists TeachesTo^- \sqsubseteq Student & \exists HasTutor^- \sqsubseteq Professor \\ Professor \sqsubseteq \neg Student & (\mathsf{funct}\ HasTutor). \end{array}$$

Assume that the ABox $\mathcal{A}$ contains only the assertion $HasTutor(\mathsf{John}, \mathsf{Mary})$. Finally, consider the query $q(x) \leftarrow TeachesTo(x, y), HasTutor(y, z)$, asking for professors that teach to students that have a tutor. ■

Although equipped with advanced reasoning services, at first sight *DL-Lite* might seem rather weak in modeling intensional knowledge, and hence of limited use in practice. In fact, this is not the case. Despite the simplicity of its language and the specific form of inclusion assertions allowed, *DL-Lite* is able to capture the main notions (though not all, obviously) of both ontologies, and of conceptual modeling formalisms used in databases and software engineering (i.e., ER and UML class diagrams). In particular, *DL-Lite* assertions allow us to specify *ISA*, e.g., stating that concept A_1 is subsumed by concept A_2, using $A_1 \sqsubseteq A_2$; *disjointness*, e.g., between concepts A_1 and A_2, using $A_1 \sqsubseteq \neg A_2$; *role-typing*, e.g., stating that the first (resp., second) component of the relation R is an instance of A_1 (resp., A_2), using $\exists R \sqsubseteq A_1$ (resp., $\exists R^- \sqsubseteq A_2$); *participation constraints*, e.g., stating that all instances of concept A participate to the relation R as the first (resp., second) component, using $A \sqsubseteq \exists R$ (resp., $A \sqsubseteq \exists R^-$); *non-participation constraints*, using $A \sqsubseteq \neg \exists R$ and $A \sqsubseteq \neg \exists R^-$; *functionality restrictions* on relations, using $(\mathsf{funct}\ R)$ and $(\mathsf{funct}\ R^-)$. Notice that DL-Lite is a strict subset of OWL Lite, the less expressive sublanguage of OWL[5], which presents some constructs (e.g., some kinds of role restrictions) that are non expressible in DL-Lite, and that make reasoning in OWL Lite non-tractable in general.

Reasoning in *DL-Lite*

It can be shown that *query containment* can be reformulated as *query answering* using techniques similar to the ones in [1]. Hence, we concentrate on query answering only.

We first address some preliminary issues, and then we define the query reformulation algorithm PerfectRef, which is at the heart of our query evaluation algorithm Answer. Finally, we address correctness and complexity issues.

[5]http://www.w3.org/TR/owl-features

KB normalization We denote by $\mathsf{Normalize}(\mathcal{K})$ the *DL-Lite* KB obtained by transforming the KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ as follows. The ABox $\mathcal{A}$ is expanded by adding to $\mathcal{A}$ the assertions $\exists R(a)$ and $\exists R^-(b)$ for each $R(a, b) \in \mathcal{A}$.

Then, assertions of $\mathcal{K}$ in which conjunctive concepts occur are rewritten by iterative application of the rule: if $B \sqsubseteq C_1 \sqcap C_2$ occurs in $\mathcal{T}$, then replace it with the two assertions $B \sqsubseteq C_1$, $B \sqsubseteq C_2$.

The TBox $\mathcal{T}$ resulting from such a transformation contains assertions of the form (i) $B_1 \sqsubseteq B_2$, where B_1 and B_2 are basic concepts (i.e., each of them is either an atomic or an existential concept), which we call *positive inclusions (PIs)*; (ii) $B_1 \sqsubseteq \neg B_2$, where B_1 and B_2 are basic concepts, which we call *negative inclusions (NIs)*; (iii) functionality assertions on roles of the form $(\mathsf{funct}\ R)$ or $(\mathsf{funct}\ R^-)$.

Then, the TBox $\mathcal{T}$ is expanded by computing all (non-trivial) NIs between basic concepts implied by $\mathcal{T}$. More precisely, the TBox $\mathcal{T}$ is closed with respect to the following inference rule: if $B_1 \sqsubseteq B_2$ occurs in $\mathcal{T}$ and either $B_2 \sqsubseteq \neg B_3$ or $B_3 \sqsubseteq \neg B_2$ occurs in $\mathcal{T}$ (where B_1, B_2, B_3 are arbitrary basic concepts), then add $B_1 \sqsubseteq \neg B_3$ to $\mathcal{T}$. It can be shown that, after the above closure of $\mathcal{T}$, for every pair of basic concepts B_1, B_2, we have that $\mathcal{T} \models B_1 \sqsubseteq \neg B_2$ iff either $B_1 \sqsubseteq \neg B_2 \in \mathcal{T}$ or $B_2 \sqsubseteq \neg B_1 \in \mathcal{T}$.

It is immediate to verify that, for every *DL-Lite* KB $\mathcal{K}$, $\mathsf{Normalize}(\mathcal{K})$ is equivalent to $\mathcal{K}$, in the sense that the set of models of $\mathcal{K}$ coincides with that of $\mathsf{Normalize}(\mathcal{K})$. In the following, without loss of generality we assume that every concept name or role name occurring in $\mathcal{A}$ also occurs in $\mathcal{T}$.

ABox storage Once the ABox is normalized, we store it under the control of a DBMS, in order to effectively manage objects in the knowledge base by means of an SQL engine. To this aim, we construct a relational database which faithfully represents a normalized ABox $\mathcal{A}$. More precisely,

- for each basic concept B occurring in $\mathcal{A}$, we define a relational table tab_B of arity 1, such that $\langle a \rangle \in tab_B$ iff $B(a) \in \mathcal{A}$;
- for each role R occurring in $\mathcal{A}$, we define a relational table tab_R of arity 2, such that $\langle a, b \rangle \in tab_R$ iff $R(a, b) \in \mathcal{A}$.

We denote with $\mathsf{DB}(\mathcal{A})$ the relational database thus constructed.

KB satisfiability The algorithm Consistent takes as input a normalized KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ and verifies the following conditions:
(i) there exists a NI $B_1 \sqsubseteq \neg B_2$ in $\mathcal{T}$ and a constant a such that the assertions $B_1(a)$ and $B_2(a)$ belong to $\mathcal{A}$;
(ii) there exists an assertion $(\mathsf{funct}\ R)$ (respectively, $(\mathsf{funct}\ R^-)$) in $\mathcal{T}$ and three constants a, b, c such that both $R(a, b)$ and $R(a, c)$ (resp., $R(b, a)$ and $R(c, a)$) belong to $\mathcal{A}$.

Informally, condition (i) corresponds to checking whether $\mathcal{A}$ explicitly contradicts some NI in $\mathcal{T}$, and condition (ii) corresponds to check whether $\mathcal{A}$ violates some functionality assertion in $\mathcal{T}$. If one of the above conditions holds, then the algorithm returns *false* (i.e., $\mathcal{K}$ is not satisfiable); otherwise, the algorithm returns *true*.

Notably, the algorithm verifies such conditions by posing to $\mathsf{DB}(\mathcal{A})$ suitable conjunctive queries expressed in SQL. For instance, condition (i) holds for a given NI $B_1 \sqsubseteq \neg B_2$ iff the query $q(x) \leftarrow tab_{B_1}(x), tab_{B_2}(x)$ has a non-empty answer in $\mathsf{DB}(\mathcal{A})$, while condition (ii) holds for (funct R) iff the query $q(x) \leftarrow tab_R(x,y), tab_R(x,z), y \neq z$ has a non-empty answer in $\mathsf{DB}(\mathcal{A})$, where $\neq$ is the "not equal" predicate of SQL. Notice that the algorithm does not consider the PIs occurring in $\mathcal{T}$ during its execution. Indeed, we will show that PIs do not affect the consistency of a *DL-Lite* KB, if the TBox is normalized.

Query reformulation Query reformulation is at the heart of our query answering method. Given the limited expressive power of *DL-Lite* TBoxes, it might seem that in order to answer a query q over a KB $\mathcal{K}$, we could simply build a finite first-order structure on the basis of $\mathcal{K}$, and then evaluate the query as an expression over this first-order structure. Actually, it is possible to show that this is not the case. In particular, it can be shown that, in general, given a KB $\mathcal{K}$, there exists no finite structure $\mathcal{S}$ such that, for every conjunctive query q, the set of answers to q over $\mathcal{K}$ is the result of evaluating q over $\mathcal{S}$. This property demonstrates that answering queries in *DL-Lite* goes beyond both propositional logic and relational databases. The basic idea of our method is to reformulate the query taking into account the TBox: in particular, given a query q over $\mathcal{K}$, we compile the assertions of the TBox into the query itself, thus obtaining a new query q'. Such a new query q' is then evaluated over the ABox of $\mathcal{K}$, as if the ABox were a simple relational database. Since the size of q' does not depend on the ABox, the data complexity of the whole query answering algorithm is polynomial. In the following, we illustrate our approach from a technical point of view.

We say that an argument of an atom in a query is *bound* if it corresponds to either a distinguished variable or a shared variable, i.e., a variable occurring at least twice in the query body, or a constant, while we say that it is *unbound* if it corresponds to a non-distinguished non-shared variable (as usual, we use the symbol _ to represent non-distinguished non-shared variables). Notice that, an atom of the form $\exists R(x)$ (resp. $\exists R^-(x)$) has the same meaning as $R(x, \_)$ (resp. $R(\_, x)$). For ease of exposition, in the following we will use the latter form only.

A PI I *is applicable to an atom* $B(x)$, if I has B in its right-hand side, and I *is applicable to an atom* $R(x_1, x_2)$, if either (i) $x_2 = \_$ and the right-hand side of I is $\exists R$, or (ii) $x_1 = \_$ and the right-hand side of I is $\exists R^-$. Roughly speaking, an inclusion I is applicable to an atom g if all bound arguments of g are propagated by I. Obviously, since all PIs in the TBox $\mathcal{T}$ are unary, they are never applicable to atoms with two bound arguments.

We indicate with $gr(g, I)$ the atom obtained from the atom g by applying the inclusion I, i.e., if $g = B_1(x)$ (resp., $g = R_1(x, \_)$ or $g = R_1(\_, x)$) and $I = B_2 \sqsubseteq B_1$ (resp., $I = B_2 \sqsubseteq \exists R_1$ or $I = B_2 \sqsubseteq \exists R_1^-$), we have:

- $gr(g, I) = R_2(x, \_)$, if $B_2 = \exists R_2$;
- $gr(g, I) = R_2(\_, x)$, if $B_2 = \exists R_2^-$;
- $gr(g, I) = A(x)$, if $B_2 = A$, where A is a basic concept.

We are now ready to define the algorithm $\mathsf{PerfectRef}$.

```
**Algorithm** PerfectRef($q, \mathcal{T}$)
**Input:** conjunctive query $q$, *DL-Lite* TBox $\mathcal{T}$
**Output:** set of conjunctive queries $P$
$P := \{q\}$;
**repeat**
  $P' := P$;
  **for each** $q \in P'$ **do**
  (a) **for each** $g$ in $q$ **do**
        **for each** PI $I$ in $\mathcal{T}$ **do**
          **if** $I$ is applicable to $g$
          **then** $P := P \cup \{\, q[g/gr(g,I)] \,\}$
  (b) **for each** $g_1, g_2$ in $q$ **do**
        **if** $g_1$ and $g_2$ unify
        **then** $P := P \cup \{\tau(reduce(q, g_1, g_2))\}$;
**until** $P' = P$;
**return** $P$
```

In the algorithm, $q[g/g']$ denotes the query obtained from q by replacing the atom g with a new atom g'.

Informally, the algorithm first reformulates the atoms of each query $q \in P'$, and produces a new query for each atom reformulation (step (a)). Roughly speaking, PIs are used as rewriting rules, applied from right to left, that allow to compile away in the reformulation the knowledge of $\mathcal{T}$ that is relevant for answering q.

At step (b), for each pair of atoms g_1, g_2 that unify, the algorithm computes the query $q' = reduce(q, g_1, g_2)$, by applying to q the *most general unifier* between g_1 and g_2. Due to the unification, variables that were bound in q may become unbound in q'. Hence, PIs that were not applicable to atoms of q, may become applicable to atoms of q' (in the next executions of step (a)). Function τ applied to q' replaces with _ each unbound variable in q'.

It can be shown that the algorithm always terminates, since the maximum number of atoms in the body of a generated query is equal to the length of the initial query, and the number of different atoms that can be generated by the algorithm is polynomial in the size of the input.

Example 1 (contd.). Let us analyze $\mathsf{PerfectRef}(q, \mathcal{T})$, where $q(x) \leftarrow TeachesTo(x,y), HasTutor(y, \_)$. At the first execution of step (a), the algorithm inserts in P the new query $q(x) \leftarrow TeachesTo(x,y), Student(y)$, by applying to the atom $HasTutor(y, \_)$ the PI $Student \sqsubseteq \exists HasTutor$. Then, at a second execution of step (a), the query $q(x) \leftarrow TeachesTo(x,y), TeachesTo(\_, y)$ is added to P, according to application of the PI $\exists TeachesTo^- \sqsubseteq Student$ to the atom $Student(y)$. Since the two atoms of the second query unify, step (b) of the algorithm inserts the query $q(x) \leftarrow TeachesTo(x, \_)$ into P. At a next iteration, step (a) produces the query $q(x) \leftarrow Professor(x)$, by applying $Professor \sqsubseteq \exists TeachesTo$ to $TeachesTo(x, \_)$, and then, at a further execution of step (a), it generates the query $q(x) \leftarrow HasTutor(\_, x)$ by applying $\exists HasTutor^- \sqsubseteq Professor$ to $Professor(x)$. The set constituted by the above five queries and the original query q is then returned by the algorithm. ■

Query evaluation In order to compute the answers to q over the KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, we need to evaluate the set of conjunctive queries P produced by the algorithm $\mathsf{PerfectRef}$ over the ABox $\mathcal{A}$. Obviously, in doing so we want to exploit the relational database $\mathsf{DB}(\mathcal{A})$. To this aim, we need to transform each query q in P into an SQL query expressed over $\mathsf{DB}(\mathcal{A})$. The transformation (which we omit for lack of space) is conceptually very simple. The only non-trivial case concerns binary atoms with unbound terms: for an atom of the form $R(\_, x)$, we introduce a view predicate that represents the union of $tab_R[2]$ with $tab_{\exists R^-}$, where $tab_R[2]$ indicates projection of tab_R on its second column (similarly for $R(x, \_)$). All SQL queries obtained from P, together with the views introduced in the transformation, denoted by $\mathsf{SQL}(P)$, can be easily dispatched to an SQL query engine and evaluated over $\mathsf{DB}(\mathcal{A})$.

Below we define the algorithm Answer that, given a satisfiable KB $\mathcal{K}$ and a query q, computes $ans(q, \mathcal{K})$[6]. In the algorithm, $\mathsf{Eval}(Q, D)$ denotes the evaluation of the SQL query Q over the database D.

```
Algorithm Answer$(q, \mathcal{K})$
Input: CQ $q$, DL-Lite KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$
Output: $ans(q, \mathcal{K})$
$\mathcal{K}$ := Normalize$(\mathcal{K})$;
return Eval(SQL(PerfectRef$(q, \mathcal{T})$), DB$(\mathcal{A})$)
```

Example 1 (contd.). Since our ABox $\mathcal{A}$ contains only the assertion $HasTutor(\mathsf{John}, \mathsf{Mary})$, it is trivial to establish satisfiability of $\mathcal{K}$ (which can be done by means of the algorithm Consistent). Then, by executing $\mathsf{Answer}(q, \mathcal{K})$, we first obtain $\mathsf{Normalize}(\mathcal{K})$, which is computed by adding to $\mathcal{T}$ all NIs implied by $\mathcal{T}$, i.e.,:

$$\exists TeachesTo^- \sqsubseteq \neg Professor \quad \exists HasTutor^- \sqsubseteq \neg Student.$$

Then, $\mathsf{Eval}(\mathsf{SQL}(\mathsf{PerfectRef}(q, \mathcal{T})), \mathsf{DB}(\mathcal{A}))$ returns the set $\{\mathsf{Mary}\}$. In particular, Mary is returned by the evaluation of the SQL transformation of the query $q(x) \leftarrow HasTutor(\_, x)$. ■

Correctness We now prove correctness of the above described query answering technique. To this aim, we use a chase-like technique [2]. Given a normalized KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, we call *chase of* $\mathcal{K}$ (denoted by $chase(\mathcal{K})$) the (possibly infinite) ABox obtained starting from $\mathcal{A}$ and closing it with respect to the following chase rules: (i) if $B_1(a) \in chase(\mathcal{K})$ and $B_1 \sqsubseteq B_2 \in \mathcal{T}$, then add $B_2(a)$ to $chase(\mathcal{K})$; (ii) if $\exists R(a) \in chase(\mathcal{K})$ (respectively, $\exists R^-(a) \in chase(\mathcal{K})$) and there exists no individual b such that $R(a, b) \in chase(\mathcal{K})$ (resp., $R(b, a) \in chase(\mathcal{K})$), then add the assertions $R(a, n)$ and $\exists R^-(n)$ (resp., $R(n, a)$ and $\exists R(n)$) to $chase(\mathcal{K})$, where n is a new constant of Δ not already occurring in $chase(\mathcal{K})$.

Intuitively, the correctness of our query processing technique is based on a crucial property of $chase(\mathcal{K})$: if $\mathcal{K}$ is satisfiable, then $chase(\mathcal{K})$ is a representative of all models of $\mathcal{K}$. This property implies that query answering can be in principle done by evaluating the query over $chase(\mathcal{K})$ seen as a database. However, since $chase(\mathcal{K})$ is in general infinite, we obviously avoid the construction of the chase. Rather, as we said before, we are able to compile the TBox into the query, thus simulating the evaluation of the query over the (in general infinite) chase by evaluating a finite reformulation of the query over the initial ABox.

We first establish correctness of the technique for deciding satisfiability of a *DL-Lite* KB.

[6]Notice that, if $\mathcal{K}$ is unsatisfiable, query answering is meaningless, since every tuple is in the answer to every query.

Theorem 2 *Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a normalized DL-Lite KB. $\mathcal{K}$ is satisfiable iff the algorithm* Consistent *returns true.*

We now establish correctness of the algoritm Answer under the assumption that the KB is satisfiable.

Theorem 3 *Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a satisfiable DL-Lite KB, let q be a CQ, and let $\vec{c}$ be a tuple of constants from Δ. Then, $\vec{c} \in ans(q, \mathcal{K})$ iff $\vec{c} \in \mathsf{Answer}(q, \mathcal{K})$.*

Complexity First, we analyze complexity of KB satisfiability in *DL-Lite*.

Theorem 4 *Satisfiability of a DL-Lite KB $\mathcal{K}$ can be decided in time polynomial in the size of $\mathcal{K}$.*

Proof (sketch). The proof immediately follows from the following facts: (i) the algorithm Normalize runs in time polynomial in the size of $\mathcal{K}$; (ii) the algorithm Consistent is correct; (iii) the algorithm Consistent runs in time polynomial in the size of the input. □

Then, from correctness of the algorithm Answer, we are immediately able to characterize complexity of conjunctive query answering in *DL-Lite* w.r.t. data complexity.

Theorem 5 *Conjunctive query answering in DL-Lite is in PTIME in data complexity.*

We are also able to characterize the combined complexity (i.e., the complexity w.r.t. the size of $\mathcal{K}$ and q) of conjunctive query answering in *DL-Lite*.

Theorem 6 *Conjunctive query answering in DL-Lite is NP-complete in combined complexity.*

Proof (sketch). Membership in NP is a consequence of the fact that, given any *DL-Lite* KB $\mathcal{K}$, if $\vec{c} \in ans(q, \mathcal{K})$, then it is possible to nondeterministically construct a *polynomial* fragment of $chase(\mathcal{K})$ which contains an image of $q(\vec{c})$. NP-hardness follows from NP-hardness of conjunctive query evaluation over relational databases. □

Finally, since in *DL-Lite* it is possibile to polynomially reduce containment between CQs to query answering, from the above results it follows that containment of conjunctive queries in *DL-Lite* is NP-complete.

Summarizing, the above results show a very nice computational behavior of queries in *DL-Lite*: reasoning in *DL-Lite* is computationally no worse than standard conjunctive query answering (and containment) in relational databases.

Discussion and related work

DL-Lite is a fragment of expressive DLs with assertions and inverses studied in the 90's (see [4] for an overview), which are at the base of current ontology languages such as OWL, and for which optimized automated reasoning systems such as Fact and Racer have been developed. Indeed, one could use, off-the-shelf, a system like Racer to perform KB satisfiability, instance checking (of concepts), and logical implication of inclusion assertions in *DL-Lite*. Also, reasoning with conjunctive queries in these DLs has been studied (see e.g. [11]), although not yet implemented in systems. Unfortunately, the reasoning procedures for these DLs are all EXPTIME-hard, and more importantly they are not tailored towards obtaining tight complexity bounds with respect to data complexity. Conjunctive queries combined with DLs were also considered in [16, 13], but again data complexity was not the main concern.

There has been a lot of work in DLs on the boundary between polynomial and exponential reasoning. This work first concentrated on DLs without the TBox component of the KB, and led to the development of simple DLs, such as $\mathcal{ALN}$, that admit polynomial instance checking. However, for minor variants of $\mathcal{ALN}$, such as $\mathcal{ALE}$ (where we introduce qualified existential and drop number restrictions), $\mathcal{FLE}^-$ (where we additionally drop negated atomic concept), and $\mathcal{ALU}$ (where we introduce union and drop number restrictions), instance checking, and therefore conjunctive query answering, is coNP-complete in data complexity [12]. Indeed, the argument used in the proof of coNP-hardness of $\mathcal{ALE}$, $\mathcal{FLE}^-$, and $\mathcal{ALU}$ in [12], immediately implies the following theorem.

Theorem 7 *Answering conjunctive queries is coNP-hard in data complexity (even in KBs with empty TBoxes), if we extend DL-Lite with one of the following features: (1) either $\forall R.A$ or $\neg A$ can appear in left-hand sides of inclusion assertions; (2) either $\forall R.A$ or $\neg A$ can appear as atoms in the query; (3) union of concepts can appear in the right-hand side of inclusion assertions.*

If we allow for cyclic inclusion assertions in the KB, then even subsumption in CLASSIC and $\mathcal{ALN}$ becomes intractable [9][7]. Observe that *DL-Lite* does allow for cyclic assertions without falling into intractability. Indeed, we can enforce the cyclic propagation of the existence of an R-successor using the two *DL-Lite* inclusion assertions $A \sqsubseteq \exists R$, $\exists R^- \sqsubseteq A$. The constraint imposed on a model is similar to the one imposed by the $\mathcal{ALN}$ cyclic assertion $A \sqsubseteq \exists R \sqcap \forall R.A$, though stronger, since it additionally enforces the second component of R to be typed by A. In order to keep tractability even in the presence of cycles, *DL-Lite* imposes restrictions on the use of the $\forall R.C$ construct, which, if used together with inclusion assertions, immediately would lead to intractability [9].

Our work is also tightly related to work in databases on implication of integrity constraints (ICs) [2] and on query answering in the presence of ICs under an open world semantics (see, e.g., [8, 3, 14, 7]). Rephrased as ICs, *DL-Lite* TBoxes allow for expressing special forms of inclusion dependencies (i.e., ISA, role typing, and participation constraints), multiple keys on relations (i.e., functionality restrictions), and exclusion dependencies (i.e., disjointness and non-participation constraints)[8]. The results that we report here show that *DL-Lite* inclusion assertions form one of the largest class of ICs for which query answering remains polynomial.

Conclusions

We have described *DL-Lite*, a new DL specifically tailored to capture conceptual data models and basic ontology languages, while keeping the worst-case complexity of sound and complete reasoning tractable.

In this paper we focused on binary roles only, but it is possible to extend our reasoning techniques to n-ary relations without loosing their nice computational properties. We are working on other interesting extensions to *DL-Lite*, such as the introduction of subset constraints on roles. The results of [10] imply that finding an adaptation of our query answering technique is going to be a hard problem.

Acknowledgments This research has been partially supported by the Projects INFOMIX (IST-2001-33570) and SEWASIE (IST-2001-34825) funded by the EU.

[7]Note that a TBox with only acyclic inclusion assertions can always be transformed into an empty TBox.

[8]Notice that this combination of ICs has only been studied in [7], but under a different semantics wrt the one adopted in DLs.

References

[1] S. Abiteboul and O. Duschka. Complexity of answering queries using materialized views. In *Proc. of PODS'98*, pages 254–265, 1998.

[2] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison Wesley Publ. Co., 1995.

[3] M. Arenas, L. E. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *Proc. of PODS'99*, pages 68–79, 1999.

[4] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.

[5] A. Borgida and R. J. Brachman. Conceptual modeling with description logics. In Baader et al. [4], chapter 10, pages 349–372.

[6] A. Borgida, R. J. Brachman, D. L. McGuinness, and L. A. Resnick. CLASSIC: A structural data model for objects. In *Proc. of ACM SIGMOD*, pages 59–67, 1989.

[7] L. Bravo and L. Bertossi. Logic programming for consistently querying data integration systems. In *Proc. of IJCAI 2003*, pages 10–15, 2003.

[8] A. Calì, D. Lembo, and R. Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *Proc. of PODS 2003*, pages 260–271, 2003.

[9] D. Calvanese. Reasoning with inclusion axioms in description logics: Algorithms and complexity. In *Proc. of ECAI'96*, pages 303–307. John Wiley & Sons, 1996.

[10] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. What to ask to a peer: Ontology-based query reformulation. In *Proc. of KR 2004*, pages 469–478, 2004.

[11] D. Calvanese, G. De Giacomo, and M. Lenzerini. Answering queries using views over description logics knowledge bases. In *Proc. of AAAI 2000*, pages 386–391, 2000.

[12] F. M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. Deduction in concept languages: From subsumption to instance checking. *J. of Log. and Comp.*, 4(4):423–452, 1994.

[13] F. M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. $\mathcal{AL}$-log: Integrating Datalog and description logics. *J. of Intelligent Information Systems*, 10(3):227–252, 1998.

[14] O. M. Duschka, M. R. Genesereth, and A. Y. Levy. Recursive query plans for data integration. *J. of Logic Programming*, 43(1):49–73, 2000.

[15] H. J. Levesque and G. Lakemeyer. *The Logic of Knowledge Bases*. The MIT Press, 2001.

[16] A. Y. Levy and M.-C. Rousset. Combining Horn rules and description logics in CARIN. *Artificial Intelligence*, 104(1–2):165–209, 1998.

[17] M. Y. Vardi. The complexity of relational query languages. In *Proc. of STOC'82*, pages 137–146, 1982.