Description Logics for Information Integration

Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini

Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, 00198 Roma, Italy
lastname@dis.uniroma1.it,
http://www.dis.uniroma1.it/$\sim$*lastname*

Abstract. Information integration is the problem of combining the data residing at different, heterogeneous sources, and providing the user with a unified view of these data, called mediated schema. The mediated schema is therefore a reconciled view of the information, which can be queried by the user. It is the task of the system to free the user from the knowledge on where data are, and how data are structured at the sources.

In this chapter, we discuss data integration in general, and describe a logic-based approach to data integration. A logic of the Description Logics family is used to model the information managed by the integration system, to formulate queries posed to the system, and to perform several types of automated reasoning supporting both the modeling, and the query answering process. We focus, in particular, on a specific Description Logic, called $\mathcal{DLR}$, specifically designed for database applications. In the chapter, we illustrate how $\mathcal{DLR}$ is used to model a mediated schema of an integration system, to specify the semantics of the data sources, and finally to support the query answering process by means of the associated reasoning methods.

1 Introduction

Information integration is the problem of combining the data residing at different sources, and providing the user with a unified view of these data, called mediated schema. The mediated schema is therefore a reconciled view of the information, which can be queried by the user. It is the task of the data integration system to free 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 different sources.

The design of a data integration system is a very complex task, which comprises several different issues, including the following:

1. heterogeneity of the sources,
2. relation between the mediated schema and the sources,
3. limitations on the mechanisms for accessing the sources,
4. materialized vs. virtual integration,
5. data cleaning and reconciliation,
6. how to process queries expressed on the mediated schema.

Problem (1) arises because sources are typically heterogeneous, meaning that they adopt different models and systems for storing data. This poses challenging problems in specifying the mediated schema. The goal is to design such a schema so as to provide an appropriate abstraction of all the data residing at the sources. One aspect deserving special attention is the choice of the language used to express the mediated schema. Since such a schema should mediate among different representations of overlapping worlds, the language should provide flexible and powerful representation mechanisms. We refer to [34] for a more detailed discussion on this subject. Following the work in [32,16,40], in this paper we use a formalism of the family of Description Logics to specify mediated schemas.

With regard to Problem (2), two basic approaches have been used to specify the relation between the sources and the mediated schema. The first approach, called *global-as-view* (or query-based), requires that the mediated schema is expressed in terms of the data sources. More precisely, to every concept of the mediated 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* (or source-based), requires the mediated schema to be specified independently from the sources. The relationships between the mediated schema and the sources are established by defining every source as a view over the mediated schema. Thus, in the local-as-view approach, we specify the meaning of the sources in terms of the concepts in the mediated schema. It is clear that the latter approach favors the 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 mediated schema. On the contrary, in the global-as-view approach, adding a new source typically requires changing the definition of the concepts in the mediated schema. For this reason, in the rest of the paper, we adopt the local-as-view approach. A comparison between the two approaches is reported in [51].

Problem (3) refers to the fact, that, both in the local-as-view and in the global-as-view approach, it may happen that a source presents some limitations on the types of accesses it supports. A typical example is a web source accessible through a form where one of the fields must necessarily be filled in by the user. Such a situation can be modeled by specifying the source as a relation supporting only queries with a selection on a column. Suitable notations have been proposed for such situations [44], and the consequences of these access limitations on query processing in integration systems have been investigated in several papers [44,43,27,56,55,41,42].

Problem (4) deals with a further criterion that we should take into account in the design of a data integration system. In particular, with respect to the

data explicitly managed by the system, we can follow two different approaches, called *materialized* and *virtual*. In the materialized approach, the system computes the extensions of the concepts in the mediated schema by replicating the data at the sources. In the virtual approach, data residing at the sources are accessed during query processing, but they are not replicated in the integration system. Obviously, in the materialized approach, the problem of refreshing the materialized views in order to keep them up-to-date is a major issue [34]. In the following, we only deal with the virtual approach.

Whereas the construction of the mediated schema concerns the intentional level of the data integration system, problem (5) refers to a number of issues arising when considering the integration at the extensional/instance level. A first issue in this context is the interpretation and merging of the data provided by the sources. Interpreting data can be regarded as the task of casting them into a common representation. Moreover, the data returned by various sources need to be converted/reconciled/combined to provide the data integration system with the requested information. The complexity of this reconciliation step is due to several problems, such as possible mismatches between data referring to the same real world object, possible errors in the data stored in the sources, or possible inconsistencies between values representing the properties of the real world objects in different sources [28]. The above task is known in the literature as *Data Cleaning and Reconciliation*, and the interested reader is referred to [28,10,4] for more details on this subject.

Finally, problem (6) is concerned with one of the most important issues in a data integration system, i.e., the choice of the method for computing the answer to queries posed in terms of the mediated schema. While query answering in the global-as-view approach typically reduces to unfolding, an integration system based on the local-as-view approach must resort to more sophisticated query processing techniques. The main issue is that the system should be able to re-express the query in terms of a suitable set of queries posed to the sources. In this reformulation process, the crucial step is deciding how to decompose the query on the mediated schema into a set of subqueries on the sources, based on the meaning of the sources in terms of the concepts in the mediated schema. The computed subqueries are then shipped to the sources, and the results are assembled into the final answer.

In the rest of this paper, we concentrate on Problem (6), namely, query processing in a data integration system specified by means of the local-as-view approach, and we present the following contributions:

– We first provide a logical formalization of the problem. In particular, we illustrate a general architecture for a data integration system, comprising a mediated schema, a set of views, and a query. Query processing in this setting is formally defined as the problem of *answering queries using views*: compute the answer to a query only on the basis of the extension of a set of views [1,29]. We observe that, besides data integration, this problem is relevant in several fields, including data warehousing [54], query optimization [17], supporting physical data independence [50], etc.

– Then we instantiate the general framework to the case where schemas, views and queries are expressed by making use of a particular logical language. In particular:
  • The mediated schema is expressed in terms of a knowledge base constituted by general inclusion assertions and membership assertions, formulated in an expressive Description Logic [6].
  • Queries and views are expressed as non-recursive datalog programs, whose predicates in the body are concepts or relations that appear in the knowledge base.
  • For each view, it can be specified whether the provided extension is sound, complete, or exact with respect to the view definition [1,11]. Such assumptions are used in data integration with the following meaning. A *sound view* corresponds to an information source which is known to produce only, but not necessarily all, the answers to the associated query. A *complete view* models a source which is known to produce all answers to the associated query, and maybe more. Finally, an *exact view* is known to produce exactly the answers to the associated query.
– We then illustrate a technique for the problem of answering queries using views in our setting. We first describe how to formulate the problem in terms of logical implication, and then we present a technique to check logical implication in 2EXPTIME worst case complexity.

The paper is organized as follows. Section 2 presents the general framework. Section 3 illustrates the use of Description Logics for setting up a particular architecture for data integration, according to the general framework. Section 4 presents the method we use for query answering using views in our architecture. Section 5 describes other works on the problem of answering query using views. Finally, Section 6 concludes the paper.

2   Framework

In this section we set up a logical framework for data integration. Since we assume to work with relational databases, in the following we refer to a relational alphabet $\mathcal{A}$, i.e., an alphabet constituted by a set of predicate and constant symbols. Predicate symbols are used to denote the relations in the database, whereas constant symbols denote the objects stored in relations. We adopt the so-called *unique name assumption*, i.e., we assume that different constants denote different objects.

A database (DB) $\mathcal{DB}$ is simply a set of relations, one for each predicate symbol in the alphabet $\mathcal{A}$. The relation corresponding to the predicate symbol R_i is constituted by a set of tuples of constants, which specify the objects that satisfy the relation associated to R_i.

The main components of a data integration system are the mediated schema, the sources, and the queries. Each component is expressed in a specific language over the alphabet $\mathcal{A}$:

- the *mediated schema* is expressed in the schema language $\mathcal{L}_\mathcal{S}$,
- the *sources* are modeled as views over the mediated schema, expressed in the view language $\mathcal{L}_\mathcal{V}$,
- *queries* are issued over the mediated schema, and are expressed in the query language $\mathcal{L}_\mathcal{Q}$.

In what follows, we provide a specification of the three components of a data integration system.

- The mediated schema $\mathcal{S}$ is a set of constraints, each one expressed in the language $\mathcal{L}_\mathcal{S}$ over the alphabet $\mathcal{A}$. The language $\mathcal{L}_\mathcal{S}$ determines the expressiveness allowed for specifying the schema of our database, i.e., the constraints that the database must satisfy. If $\mathcal{S}$ is constituted by the constraints $\{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$, we say that a database $\mathcal{DB}$ satisfies $\mathcal{S}$ if all constraints $\mathcal{C}_1, \ldots, \mathcal{C}_n$ are satisfied by $\mathcal{DB}$.
- The sources are modeled in terms of a set of views $\mathcal{V} = \{V_1, \ldots, V_m\}$ over the mediated schema. Associated to each view V_i we have:
  - A definition $def(V_i)$ in terms of a query $V_i(\boldsymbol{x}) \leftarrow v_i(\boldsymbol{x}, \boldsymbol{y})$ over $\mathcal{DB}$, where $v_i(\boldsymbol{x}, \boldsymbol{y})$ is expressed in the language $\mathcal{L}_\mathcal{V}$ over the alphabet $\mathcal{A}$. The arity of $\boldsymbol{x}$ determines the arity of the view V_i.
  - A set $ext(V_i)$ of tuples of constants, which provides the information about the extension of V_i, i.e., the content of the sources. The arity of each tuple is the same as that of V_i.
  - A specification $as(V_i)$ of which *assumption* to adopt for the view V_i, i.e., how to interpret the content of the source $ext(V_i)$ with respect to the actual set of tuples in $\mathcal{DB}$ that satisfy V_i. We describe below the various possibilities that we consider for $as(V_i)$.
- A query is expressed in the language $\mathcal{L}_\mathcal{Q}$ over the alphabet $\mathcal{A}$, and is intended to provide the specification of which data to extract from the virtual database represented in the integration system. In general, if Q is a query and $\mathcal{DB}$ is a database satsfying $\mathcal{S}$, we denote with $ans(Q, \mathcal{DB})$ the set of tuples in $\mathcal{DB}$ that satisfy Q.

The specification $as(V_i)$ determines how accurate is the knowledge on the pairs satisfying the views, i.e., how accurate is the source with respect to the specification $def(V_i)$[1]. As pointed out in several papers [1,29,37,11], the following three assumptions are relevant in a data integration system:

- *Sound Views.* When a view V_i is *sound* (denoted with $as(V_i) = sound$), its extension provides any subset of the tuples satisfying the corresponding definition. In other words, from the fact that a tuple is in $ext(V_i)$ one can conclude that it satisfies the view, while from the fact that a tuple is not in $ext(V_i)$ one cannot conclude that it does not satisfy the view. Formally, a database $\mathcal{DB}$ is *coherent with the sound view* V_i, if $ext(V_i) \subseteq ans(def(V_i), \mathcal{DB})$.

---

[1] In some papers, for example [11], different assumptions on the domain of the database are also taken into account.

- *Complete Views.* When a view V_i is *complete* (denoted with $as(V_i) = complete$), its extension provides any superset of the tuples satisfying the corresponding definition. In other words, from the fact that a tuple is in $ext(V_i)$ one cannot conclude that such a tuple satisfies the view. On the other hand, from the fact that a tuple is not in $ext(V_i)$ one can conclude that such a tuple does not satisfy the view. Formally, a database $\mathcal{DB}$ is *coherent with the complete view* V_i, if $ext(V_i) \supseteq ans(def(V_i), \mathcal{DB})$.
- *Exact Views.* When a view V_i is *exact* (denoted with $as(V_i) = exact$), its extension is exactly the set of tuples of objects satisfying the corresponding definition. Formally, a database $\mathcal{DB}$ is *coherent with the exact view* V_i, if $ext(V_i) = ans(def(V_i), \mathcal{DB})$.

The ultimate goal of a data integration system is to allow a client to extract information from the database, taking into account that the only knowledge s/he has on the database is the extension of the set of views, i.e., the content of the sources. More precisely, the problem of extracting information from the data integration system reduces to the problem of *answering queries using views*. Given

- a schema $\mathcal{S}$,
- a set of views $\mathcal{V} = \{V_1, \ldots, V_m\}$, with, for each V_i,
   - its definition $def(V_i)$,
   - its extension $ext(V_i)$, and
   - the specification $as(V_i)$ of whether it is sound, complete, or exact,
- a query Q of arity n, and
- a tuple $\boldsymbol{d} = (d_1, \ldots, d_n)$ of constants,

the problem consists in deciding whether $\boldsymbol{d} \in ans(Q, \mathcal{S}, \mathcal{V})$, i.e., deciding whether $(d_1, \ldots, d_n) \in ans(Q, \mathcal{DB})$, for each $\mathcal{DB}$ such that:

- $\mathcal{DB}$ satisfies the schema $\mathcal{S}$,
- $\mathcal{DB}$ is coherent with $V_1, \ldots, V_m$.

¿From the above definition, it is easy to see that answering queries using views is essentially an extended form of reasoning in the presence of incomplete information [53]. Indeed, when we answer the query on the basis of the views, we know only the extensions of the views, and this provides us with only partial information on the database. Moreover, since the query language may admit various forms of incomplete information (due to union, for instance), there are in general several possible databases that are coherent with the views.

The following example rephrases an example given in [1].

Example 1. Consider a relational alphabet containing (among other symbols) a binary predicate couple, and two constants Ann and Bill. Consider also two views female and male, respectively with definitions

$$\mathsf{female}(f) \leftarrow \mathsf{couple}(f, m)$$
$$\mathsf{male}(m) \leftarrow \mathsf{couple}(f, m)$$

and extensions $ext(\mathsf{female}) = \{\mathsf{Ann}\}$ and $ext(\mathsf{male}) = \{\mathsf{Bill}\}$, and assume that there are no constraints imposed by a schema.

If both views are sound, we only know that some couple has Ann as its female component and Bill as its male component. Therefore, the query $Q_c(x, y) \leftarrow \mathsf{couple}(x, y)$ asking for all couples would return an empty answer, i.e., $ans(Q_c, \mathcal{S}, \mathcal{V}) = \emptyset$. However, if both views 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., $ans(Q_c, \mathcal{S}, \mathcal{V}) = (\mathsf{Ann}, \mathsf{Bill})$. ∎

3   Specifying the Content of the Data Integration System

We propose here an architecture for data integration that is coherent with the framework described in Section 2, and is based on Description Logics [9,8]. In such an architecture, to specify mediated schemas, views, and queries we use the Description Logic $\mathcal{DLR}$ [6]. We first introduce $\mathcal{DLR}$, and then we illustrate how we use the logic to specify the three components of a data integration system.

3.1   The Description Logic $\mathcal{DLR}$

Description Logics[2] (DLs) have been introduced in the early 80's in the attempt to provide a formal ground to Semantic Networks and Frames. Since then they have evolved into knowledge representation languages that are able to capture virtually all class-based representation formalisms used in Artificial Intelligence, Software Engineering, and Databases. One of the distinguishing features of the work on these logics is the detailed computational complexity analysis both of the associated reasoning algorithms, and of the logical implication problem that the algorithms are supposed to solve. By virtue of this analysis, most of these logics have optimal reasoning algorithms, and practical systems implementing such algorithms are now used in several projects. In DLs, the domain of interest is modeled by means of *concepts* and *relations*, which denote classes of objects and relationships, respectively.

Here, we focus our attention on the DL $\mathcal{DLR}$ [5,6]. The basic elements of $\mathcal{DLR}$ are *concepts* (unary relations), and *n-ary relations*. We assume to deal with an alphabet $\mathcal{A}$ constituted by a finite set of atomic relations, atomic concepts, and *constants*, denoted by P, A, and a, respectively. We use R to denote arbitrary relations (of given arity between 2 and n_{max}), and C to denote arbitrary concepts, respectively built according to the following syntax:

$$R ::= \top_n \mid P \mid \$i/n{:}C \mid \neg R \mid R_1 \sqcap R_2$$
$$C ::= \top_1 \mid A \mid \neg C \mid C_1 \sqcap C_2 \mid \exists[\$i]R \mid (\leq k\,[\$i]R)$$

where i denotes a component of a relation, i.e., an integer between 1 and n_{max}, n denotes the *arity* of a relation, i.e., an integer between 2 and n_{max}, and k denotes a nonnegative integer. We also use the following abbreviations:

---

[2] See http://dl.kr.org for the home page of Description Logics.

$$\top_n^{\mathcal{I}} \subseteq (\Delta^{\mathcal{I}})^n$$
$$P^{\mathcal{I}} \subseteq \top_n^{\mathcal{I}}$$
$$\$i/n : C^{\mathcal{I}} = \{(d_1, \ldots, d_n) \in \top_n^{\mathcal{I}} \mid d_i \in C^{\mathcal{I}}\}$$
$$(\neg R)^{\mathcal{I}} = \top_n^{\mathcal{I}} \setminus R^{\mathcal{I}}$$
$$(R_1 \sqcap R_2)^{\mathcal{I}} = R_1^{\mathcal{I}} \cap R_2^{\mathcal{I}}$$

$$\top_1^{\mathcal{I}} = \Delta^{\mathcal{I}}$$
$$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$$
$$(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$$
$$(C_1 \sqcap C_2)^{\mathcal{I}} = C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$$
$$(\exists [\$i] R)^{\mathcal{I}} = \{d \in \Delta^{\mathcal{I}} \mid \exists (d_1, \ldots, d_n) \in R^{\mathcal{I}} . d_i = d\}$$
$$(\leq k\, [\$i] R)^{\mathcal{I}} = \{d \in \Delta^{\mathcal{I}} \mid \sharp\{(d_1, \ldots, d_n) \in R^{\mathcal{I}} \mid d_i = d\} \leq k\}$$

Fig. 1. Semantic rules for $\mathcal{DLR}$ (P, R, R_1, and R_2 have arity n)

- $\bot$ for $\neg \top$,
- $C_1 \sqcup C_2$ for $\neg(\neg C_1 \sqcap \neg C_2)$,
- $C_1 \Rightarrow C_2$ for $\neg C_1 \sqcup C_2$, and
- $C_1 \equiv C_2$ for $(C_1 \Rightarrow C_2) \sqcap (C_2 \Rightarrow C_1)$.

We consider only concepts and relations that are *well-typed*, which means that

- only relations of the same arity n are combined to form expressions of type $R_1 \sqcap R_2$ (which inherit the arity n), and
- $i \leq n$ whenever i denotes a component of a relation of arity n.

The semantics of $\mathcal{DLR}$ is specified as follows. An *interpretation* $\mathcal{I}$ is constituted by an *interpretation domain* $\Delta^{\mathcal{I}}$, and an *interpretation function* $\cdot^{\mathcal{I}}$ that assigns to each constant an element of $\Delta^{\mathcal{I}}$ under the unique name assumption, to each concept C a subset $C^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$, and to each relation R of arity n a subset $R^{\mathcal{I}}$ of $(\Delta^{\mathcal{I}})^n$, such that the conditions in Figure 1 are satisfied. Observe that, the "$\neg$" constructor on relations is used to express difference of relations, and not the complement [6].

3.2   Mediated Schema, Views, and Queries

We remind the reader that a mediated schema is constituted by a finite set of constraints expressed in a schema language $\mathcal{L}_{\mathcal{S}}$. In our setting, the schema language $\mathcal{L}_{\mathcal{S}}$ is based on the DL $\mathcal{DLR}$. In particular, each constraint is formulated as an *assertion* of one of the following forms:

$$R_1 \sqsubseteq R_2 \qquad\qquad\qquad C_1 \sqsubseteq C_2$$

where R_1 and R_2 are $\mathcal{DLR}$ relations of the same arity, and C_1 and C_2 are $\mathcal{DLR}$ concepts.

As we said before, a database $\mathcal{DB}$ is a set of relations, one for each predicate symbol in the alphabet $\mathcal{A}$. We denote with $R^{\mathcal{DB}}$ the relation in $\mathcal{DB}$ corresponding

to the predicate symbol R (either an atomic concept, or an atomic relation). Note that a database can be seen as an interpretation for $\mathcal{DLR}$, whose domain coincides with the set of constants in the alphabet $\mathcal{A}$.

We say that a database $\mathcal{DB}$ *satisfies* an assertion $R_1 \sqsubseteq R_2$ (resp., $C_1 \sqsubseteq C_2$) if $R_1^{\mathcal{DB}} \subseteq R_2^{\mathcal{DB}}$ (resp., $C_1^{\mathcal{DB}} \subseteq C_2^{\mathcal{DB}}$). Moreover, $\mathcal{DB}$ satisfies a schema $\mathcal{S}$ if $\mathcal{DB}$ satisfies all assertions in $\mathcal{S}$.

In order to define views and queries, we now introduce the notion of query expression in our setting. We assume that the alphabet $\mathcal{A}$ is enriched with a finite set of variable symbols, simply called variables.

A *query expression* Q is a non-recursive datalog query of the form

$$Q(\boldsymbol{x}) \;\leftarrow\; conj_1(\boldsymbol{x}, \boldsymbol{y}_1) \vee \cdots \vee conj_m(\boldsymbol{x}, \boldsymbol{y}_m)$$

where each $conj_i(\boldsymbol{x}, \boldsymbol{y}_i)$ is a conjunction of *atoms*, and $\boldsymbol{x}, \boldsymbol{y}_i$ are all the variables appearing in the conjunct. Each atom has one of the forms $R(\boldsymbol{t})$ or $C(t)$, where $\boldsymbol{t}$ and t are variables in $\boldsymbol{x}$ and $\boldsymbol{y}_i$ or constants in $\mathcal{A}$, R is a relation, and C is a concept. The number of variables of $\boldsymbol{x}$ is called the *arity* of Q, and is the arity of the relation denoted by the query Q.

We observe that the atoms in the query expressions are arbitrary $\mathcal{DLR}$ relations and concepts, freely used in the assertions of the KB. This distinguishes our approach with respect to [22,39], where no constraints on the relations that appear in the queries can be expressed in the KB.

Given a database $\mathcal{DB}$, a query expression Q of arity n is interpreted as the set $Q^{\mathcal{DB}}$ of n-tuples of constants $(c_1, \ldots, c_n)$, such that, when substituting each c_i for x_i, the formula

$$\exists \boldsymbol{y}_1.conj_1(\boldsymbol{x}, \boldsymbol{y}_1) \vee \cdots \vee \exists \boldsymbol{y}_m.conj_m(\boldsymbol{x}, \boldsymbol{y}_m)$$

evaluates to true in $\mathcal{DB}$.

With the introduction of query expressions, we can now define views and queries. Indeed, in our setting, query expressions constitute both the view language $\mathcal{L}_{\mathcal{V}}$, and the query language $\mathcal{L}_{\mathcal{Q}}$:

- Associated to each view V_i in the set $\mathcal{V} = \{V_1, \ldots, V_m\}$ we have:
    - A definition $def(V_i)$ in terms of a query expression
    - A set $ext(V_i)$ of tuples of constants,
    - A specification $as(V_i)$ of which *assumption* to adopt for the view V_i, where each $as(V_i)$ is either *sound*, *complete*, or *exact*.
- A query is simply a query expression, as defined above.

Example 2. Consider for example the following $\mathcal{DLR}$ schema $\mathcal{S}_d$, expressing that Americans who have a doctor as relative are wealthy, and that each surgeon is also a doctor

$$\mathsf{American} \sqcap \exists[\$1](\mathsf{RELATIVE} \sqcap \$2 : \mathsf{Doctor}) \sqsubseteq \mathsf{Wealthy}$$
$$\mathsf{Surgeon} \sqsubseteq \mathsf{Doctor}$$

and two sound views V_1 and V_2, respectively with definitions

$$V_1(x) \leftarrow \mathsf{RELATIVE}(x, y) \wedge \mathsf{Surgeon}(y)$$
$$V_2(x) \leftarrow \mathsf{American}(x)$$

and extensions

$$ext(V_1) = \{\mathsf{Ann}, \mathsf{Bill}\}$$
$$ext(V_2) = \{\mathsf{Ann}, \mathsf{Dan}\}$$

Given the query $Q_w(x) \leftarrow \mathsf{Wealthy}(x)$, asking for those who are wealthy, we have that the only constant in $ans(Q_w, \mathcal{S}_d, \mathcal{V})$ is Ann. Moreover, if we add an exact view V_3 with definition $V_3(x) \leftarrow \mathsf{Wealthy}(x)$, and an extension $ext(V_3)$ not containing Bill, then, from the constraints in $\mathcal{S}_d$ and the information we have on the views, we can conclude that Bill is not American.   ∎

3.3   Discussion

We observe that $\mathcal{DLR}$ is able to capture a great variety of data models with many forms of constraints [15,6]. For example, $\mathcal{DLR}$ is capable to capture formally Conceptual Data Models typically used in databases [33,24], such as the Entity-Relationship Model [18]. Hence, in our setting, query answering using views is done under the constraints imposed by a conceptual data model.

The interest in $\mathcal{DLR}$ is not confined to the expressiveness it provides for specifying data schemas. It is also equipped with effective reasoning techniques that are sound and complete with respect to the semantics. In particular, checking whether a given assertion logically follows from a set of assertions is EXPTIME-complete in $\mathcal{DLR}$ (assuming that numbers are encoded in unary), and query containment, i.e., checking whether one query is contained in another one in every model of a set of assertions, is EXPTIME-hard and solvable in 2EXP-TIME [6].

4   Query Answering

In this section we study the problem of query answering using views in the setting just defined: the schema is expressed as a $\mathcal{DLR}$ knowledge base, and queries and view definitions are espressed as $\mathcal{DLR}$ query expressions. We call the resulting problem *answering query using views in $\mathcal{DLR}$*. The technical results regarding answering query using views in $\mathcal{DLR}$ illustrated in this section are taken from [7].

The first thing to observe is that, given a schema $\mathcal{S}$ expressed in $\mathcal{DLR}$, a set of views $\mathcal{V} = \{V_1, \ldots, V_m\}$, a query Q, and a tuple $\boldsymbol{d} = (d_1, \ldots, d_n)$ of constants, verifying whether, $\boldsymbol{d}$ is in $ans(Q, \mathcal{S}, \mathcal{V})$ is essentially a form of logical implication. This observation can be made even sharper if we introduce special assertions, expressed in first-order logic with equality, that encode as logical formulas the extension of the views. In particular, for each *view $V \in \mathcal{V}$*, with $def(V) = (V(\boldsymbol{x}) \leftarrow v(\boldsymbol{x}, \boldsymbol{y}))$ and $ext(V) = \{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_k\}$, we introduce the following assertions.

- If V is *sound*, then for each tuple $\boldsymbol{a}_i$, $1 \leq i \leq k$, we introduce the existentially quantified assertion

$$\exists \boldsymbol{y}.v(\boldsymbol{a}_i, \boldsymbol{y})$$

- If V is *complete*, then we introduce the universally quantified assertion

$$\forall \boldsymbol{x}.\forall \boldsymbol{y}.((\boldsymbol{x} \neq \boldsymbol{a}_1 \wedge \cdots \wedge \boldsymbol{x} \neq \boldsymbol{a}_k) \rightarrow \neg v(\boldsymbol{x}, \boldsymbol{y}))$$

- If V is *exact*, then, according to the definition, we treat it as a view that is both sound and complete, and introduce both types of assertions above.

Let us call $Ext(\mathcal{V})$ the set of assertions corresponding to the extension of the views $\mathcal{V}$.

Now, the problem of query answering using views in $\mathcal{DLR}$, i.e., checking whether $\boldsymbol{d} \in ans(Q, \mathcal{S}, \mathcal{V})$, can be reformulated as checking whether the following logical implication holds:

$$\mathcal{S} \cup Ext(\mathcal{V}) \models \exists \boldsymbol{y}.q(\boldsymbol{d}, \boldsymbol{y})$$

where $q(\boldsymbol{x}, \boldsymbol{y})$ is the right hand part of Q. Checking such a logical implication can in turn be rephrased as checking the unsatisfiability of

$$\mathcal{S} \ \cup \ Ext(\mathcal{V}) \ \cup \ \{\forall \boldsymbol{y}.\neg q(\boldsymbol{d}, \boldsymbol{y})\}$$

Observe that the assertion $\forall \boldsymbol{y}.\neg q(\boldsymbol{d}, \boldsymbol{y})$ has the same form as the universal assertion used for expressing extensions of complete views, except that the antecedent in the implication is empty.

The problem with the newly introduced assertions is that they are not yet expressed in a DL. The next step is to translate them in a DL. Instead of working directly with $\mathcal{DLR}$, we are going to translate the problem of query answering using views in $\mathcal{DLR}$ to reasoning in a DL, called $\mathcal{CIQ}$, that directly corresponds to a variant of Propositional Dynamic Logic [20,6].

4.1   The Description Logic $\mathcal{CIQ}$

The DL $\mathcal{CIQ}$ is obtained from $\mathcal{DLR}$ by restricting relations to be binary (such relations are called *roles* and *inverse roles*) and allowing for complex roles corresponding to regular expressions [20].

Concepts of $\mathcal{CIQ}$ are formed according to the following abstract syntax:

$$
\begin{aligned}
C &::= \top \ \mid \ A \ \mid \ C_1 \sqcap C_2 \ \mid \ \neg C \ \mid \ \exists R.C \ \mid \ (\leq k\, Q.\, C) \\
Q &::= P \ \mid \ P^- \\
R &::= Q \ \mid \ R_1 \sqcup R_2 \ \mid \ R_1 \circ R_2 \ \mid \ R^* \ \mid \ R^- \ \mid \ id(C)
\end{aligned}
$$

where A denotes an atomic concept, C a generic concept, P an atomic role, Q a *simple role*, i.e., either an atomic role or the inverse of an atomic role, and R a generic role. We also use the following abbreviations:

$$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$$
$$\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$$
$$(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$$
$$(C_1 \sqcap C_2)^{\mathcal{I}} = C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$$
$$(\exists R.C)^{\mathcal{I}} = \{d \in \Delta^{\mathcal{I}} \mid \exists (d, d') \in R^{\mathcal{I}}. d' \in C^{\mathcal{I}}\}$$
$$(\leq k\,Q.\,C)^{\mathcal{I}} = \{d \in \Delta^{\mathcal{I}} \mid \sharp\{(d, d') \in Q^{\mathcal{I}} \mid d' \in C^{\mathcal{I}}\} \leq k\}$$

$$P^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$$
$$(R_1 \sqcup R_2)^{\mathcal{I}} = R_1^{\mathcal{I}} \cup R_2^{\mathcal{I}}$$
$$(R_1 \circ R_2)^{\mathcal{I}} = R_1^{\mathcal{I}} \circ R_2^{\mathcal{I}}$$
$$(R^*)^{\mathcal{I}} = (R^{\mathcal{I}})^* = \bigcup_{i \geq 0} (R^{\mathcal{I}})^i$$
$$(R^-)^{\mathcal{I}} = \{(d_1, d_2) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid (d_2, d_1) \in R^{\mathcal{I}}\}$$
$$id(C)^{\mathcal{I}} = \{(d, d) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid d \in C^{\mathcal{I}}\}$$

Fig. 2. Semantic rules for $\mathcal{CIQ}$

  – $\forall R.C$ for $\neg \exists R.\neg C$,
  – $(\geq k\,Q.\,C)$ for $\neg(\leq k{-}1\,Q.\,C)$

The semantic conditions for $\mathcal{CIQ}$ are specified in Figure 2 [3].

The use of $\mathcal{CIQ}$ allows us to exploit various results established recently for reasoning in such a logic. The basis of these results lies in the correspondence between $\mathcal{CIQ}$ and a variant of Propositional Dynamic Logic [26,35] that includes converse programs and "graded modalities" [25,52] on atomic programs and their converse [47]. $\mathcal{CIQ}$ inherits from Propositional Dynamic Logics the ability of *internalizing* assertions. Indeed, one can define a role U that essentially corresponds to a *universal modality*, as the reflexive-transitive closure of all roles and inverse roles in the language. Using such a universal modality we can re-express each assertion $C_1 \sqsubseteq C_2$ as the concept $\forall U.(C_1 \Rightarrow C_2)$. This allows us to re-express logical implication as concept satisfiability [47]. Concept satisfiability (and hence logical implication) in $\mathcal{CIQ}$ is EXPTIME-complete [20].

Although $\mathcal{CIQ}$ does not have constructs for *n*-ary relations as $\mathcal{DLR}$, it is possible to represent *n*-ary relations in a *sound and complete* way wrt concept satisfiability (and hence logical implication) by means of *reification* [20]. An atomic relation P is reified by introducing a new atomic concept A_P and n functional roles $f_1, \ldots, f_n$, one for each component of P. In this way, a tuple of the relation is represented by an instance of the corresponding concept, which is linked through each of the associated roles to an object representing the component of the tuple. Performing the reification requires however some attention, since in a relation there may not be two equal tuples (i.e., constituted by the same components in the same positions) in its extension. In the reified counterpart, on the other hand, one cannot explicitly rule out (e.g., by using specific assertions) that there are two objects o_1 and o_2 "representing" the same tuple, i.e., that are connected to exactly the same objects denoting the components of

---

[3] The notation $(R^{\mathcal{I}})^i$ stands for i repetitions of $R^{\mathcal{I}}$ – i.e., $(R^{\mathcal{I}})^1 = R^{\mathcal{I}}$, and $(R^{\mathcal{I}})^i = R^{\mathcal{I}} \circ (R^{\mathcal{I}})^{i-1}$.

the tuple. However, due to the fundamental inability of $\mathcal{CIQ}$ to express that two role sequences meet in the same object, no $\mathcal{CIQ}$ concept can force such a situation. Therefore one does not need to take this constraint explicitly into account when reasoning.

Finally, we are going to make use of $\mathcal{CIQ}$ extended with object-names. An *object-name* is an atomic concept that, in each model, has as extension a single object. Object-names are not required to be disjoint, i.e, we do not make the unique name assumption on them. Disjointness can be explicitly enforced when needed through explicit assertions. In general, adding object-names to $\mathcal{CIQ}$ makes reasoning NEXPTIME-hard [49]. However our use of object-names in $\mathcal{CIQ}$ is restricted so as to keep reasoning in EXPTIME.

4.2   Reduction of Answering Queries Using Views in $\mathcal{DLR}$ to $\mathcal{CIQ}$ Unsatisfiability

We tackle answering queries using views in $\mathcal{DLR}$, by reducing the problem of checking whether $\boldsymbol{d} \in ans(Q, \mathcal{S}, \mathcal{V})$ to the problem of checking the unsatisfiability of a $\mathcal{CIQ}$ concept in which object-names appear. Object-names are then eliminated, thus obtaining a $\mathcal{CIQ}$ concept.

We translate $\mathcal{S} \cup Ext(\mathcal{V})$ into a $\mathcal{CIQ}$ concept as follows. First, we eliminate n-ary relations by means of *reification*. Then, we reformulate each assertion in $\mathcal{S}$ as a concept by internalizing assertions. Instead, representing assertions in $Ext(\mathcal{V})$ requires the following ad-hoc techniques.

We translate each existentially quantified assertion

$$\exists \boldsymbol{y}.v(\boldsymbol{a}, \boldsymbol{y})$$

as follows. We represent every constant a_i by an object-name N_{a_i}, enforcing disjointness between the object-names corresponding to different constants. We represent each existentially quantified variable y, treated as a Skolem constant, by a new object-name without disjointness constraints. We also use additional concept-names representing tuples of objects. Specifically:

- An atom $C(t)$, where C is a concept and t is a term (either a constant or a variable), is translated to

$$\forall U.(N_t \Rightarrow \sigma(C))$$

  where $\sigma(C)$ is the reified counterpart of C, N_t is the object-name corresponding to t, and U is the reflexive-transitive closure of all roles and inverse roles introduced in the reification.
- An atom $R(\boldsymbol{t})$, where R is a relation of arity n and $\boldsymbol{t} = (t_1, \ldots, t_n)$ is a tuple of terms, is translated to the conjunction of the following concepts:

$$\forall U.(N_{\boldsymbol{t}} \Rightarrow \sigma(R))$$

  where $\sigma(R)$ is the reified counterpart of R and $N_{\boldsymbol{t}}$ is an object-name corresponding to $\boldsymbol{t}$,

$$\forall U.(N_{\boldsymbol{t}} \equiv (\exists f_1.N_{t_1} \sqcap \cdots \sqcap \exists f_n.N_{t_n}))$$

and for each i, $1 \leq i \leq n$, a concept

$$\forall U.(N_{t_i} \Rightarrow ((\exists f_i^-.N_t) \sqcap (\leq 1 f_i^-.N_t)))$$

Then, the translations of the atoms are combined as in $v(\boldsymbol{a}, \boldsymbol{y})$.

To translate universally quantified assertions corresponding to the complete views and also to the query, it is sufficient to deal with assertions of the form:

$$\forall \boldsymbol{x}.\forall \boldsymbol{y}.((\boldsymbol{x} \neq \boldsymbol{a}_1 \wedge \cdots \wedge \boldsymbol{x} \neq \boldsymbol{a}_k) \rightarrow \neg conj(\boldsymbol{x}, \boldsymbol{y}))$$

Following [6], we construct for $conj(\boldsymbol{x}, \boldsymbol{y})$ a special graph, called *tuple-graph*, which reflects the dependencies between variables. Specifically, the tuple-graph is used to detect cyclic dependencies. In general, the tuple-graph is composed of $\ell \geq 1$ connected components. For the i-th connected component we build a $\mathcal{CIQ}$ concept $\delta_i(\boldsymbol{x}, \boldsymbol{y})$ as in [6]. Such a concept contains newly introduced concepts A_x and A_y, one for each x in $\boldsymbol{x}$ and y in $\boldsymbol{y}$. We have to treat variables in $\boldsymbol{x}$ and $\boldsymbol{y}$ that occur in a cycle in the tuple-graph differently from those outside of cycles. Let $\boldsymbol{x}_c$ (resp., $\boldsymbol{y}_c$) denote the variables in $\boldsymbol{x}$ (resp., $\boldsymbol{y}$) that occur in a cycle, and $\boldsymbol{x}_l$ (resp., $\boldsymbol{y}_l$) those that do not occur in cycles. We first define the concept

$$C[\boldsymbol{x}_c/\boldsymbol{s}, \boldsymbol{y}_c/\boldsymbol{t}]$$

as the concept obtained from

$$(\forall U.\neg \delta_1(\boldsymbol{x}, \boldsymbol{y})) \sqcup \cdots \sqcup (\forall U.\neg \delta_\ell(\boldsymbol{x}, \boldsymbol{y}))$$

as follows:

- for each variable x_i in $\boldsymbol{x}_c$ (resp., y_i in $\boldsymbol{y}_c$), the concept A_{x_i} (resp., A_{y_i}) is replaced by N_{s_i} (resp., N_{t_i});
- for each variable y_i in $\boldsymbol{y}_l$, the concept A_{y_i} is replaced by $\top$.

Then the concept corresponding to the universally quantified assertion is constructed as the conjunction of:

- $\forall U.C_{\boldsymbol{x}_l}$, where $C_{\boldsymbol{x}_l}$ is obtained from $\boldsymbol{x} \neq \boldsymbol{a}_1 \wedge \cdots \wedge \boldsymbol{x} \neq \boldsymbol{a}_k$ by replacing each $(x \neq a)$ with $(A_x \equiv \neg N_a)$. Observe that $(x_1, \ldots, x_n) \neq (a_1, \ldots, a_n)$ is an abbreviation for $(x_1 \neq a_1 \vee \cdots \vee x_n \neq a_n)$.
- One concept $C[\boldsymbol{x}_c/\boldsymbol{s}, \boldsymbol{y}_c/\boldsymbol{t}]$ for each possible instantiation of $\boldsymbol{s}$ and $\boldsymbol{t}$ with the constants in $Ext(\mathcal{V}) \cup \{\boldsymbol{d}\}$, with the proviso that $\boldsymbol{s}$ cannot coincide with any of the $\boldsymbol{a}_i$, for $1 \leq i \leq k$ (notice that the proviso applies only in the case where all variables in $\boldsymbol{x}$ occur in a cycle in the tuple-graph).

The critical point in the above construction is how to express a universally quantified assertion

$$\forall \boldsymbol{x}.\forall \boldsymbol{y}.((\boldsymbol{x} \neq \boldsymbol{a}_1 \wedge \cdots \wedge \boldsymbol{x} \neq \boldsymbol{a}_k) \rightarrow \neg conj(\boldsymbol{x}, \boldsymbol{y}))$$

If there are no cycles in the corresponding tuple-graph, then we can directly translate the assertion into a $\mathcal{CIQ}$ concept. As shown in the construction above,

dealing with a nonempty antecedent requires some special care to correctly encode the exceptions to the universal rule. Instead, if there is a cycle, due to the fundamental inability of $\mathcal{CIQ}$ to express that two role sequences meet in the same object, no $\mathcal{CIQ}$ concept can directly express the universal assertion. The same inability, however, is shared by $\mathcal{DLR}$. Hence we can assume that the only cycles present in a model are those formed by the constants in the extension of the views or those in the tuple for which we are checking whether it is a certain answer of the query. And these are taken care of by the explicit instantiation.

As the last step to obtain a $\mathcal{CIQ}$ concept, we need to encode object-names in $\mathcal{CIQ}$. To do so we can exploit the construction used in [21] to encode $\mathcal{CIQ}$-ABoxes as concepts. Such a construction applies to the current case without any need of major adaptation. It is crucial to observe that the translation above uses object-names in order to form a sort of disjunction of ABoxes (cfr. [31]).

In [7], the following basic fact is proved for the construction presented above. Let C_{qa} be the $\mathcal{CIQ}$ concept obtained by the construction above. Then $\boldsymbol{d} \in ans(Q, \mathcal{S}, \mathcal{V})$ if and only if C_{qa} is unsatisfiable.

The size of C_{qa} is polynomial in the size of the query, of the view definitions, and of the inclusion assertions in $\mathcal{S}$, and is at most exponential in the number of constants in $ext(\mathcal{V}) \cup \{\boldsymbol{d}\}$. The exponential blow-up is due to the number of instantiations of $C[\boldsymbol{x}_c/\boldsymbol{s}, \boldsymbol{y}_c/\boldsymbol{t}]$ with constants in $ext(\mathcal{V}) \cup \{\boldsymbol{d}\}$ that are needed to capture universally quantified assertions. Hence, considering EXPTIME-completeness of satisfiability in $\mathcal{DLR}$ and in $\mathcal{CIQ}$, we get that query answering using views in $\mathcal{DLR}$ is EXPTIME-hard and can be done in 2EXPTIME.

5   Related Work

We already observed that query answering using views can be seen as a form of reasoning with incomplete information. The interested reader is referred to [53] for a survey on this subject.

We also observe that, to compute the whole set $ans(Q, \mathcal{S}, \mathcal{V})$, we need to run the algorithm presented above once for each possible tuple (of the arity of Q) of objects in the view extensions. Since we are dealing with incomplete information in a rich language, we should not expect to do much better than considering each tuple of objects separately. Indeed, in such a setting reasoning on objects, such as query answering, requires sophisticated forms of logical inference. In particular, verifying whether a certain tuple belongs to a query gives rise to a line of reasoning which may depend on the tuple under consideration, and which may vary substantially from one tuple to another. For simple languages we may indeed avoid considering tuples individually, as shown in [45] for query answering in the DL $\mathcal{ALN}$ without cyclic TBox assertions. Observe, however, that for such a DL, reasoning on objects is polynomial in both data and expression complexity [36,46], and does not require sophisticated forms of inference.

Query answering using views has been investigated in the last years in the context of simplified frameworks. In [38,44], the problem has been studied for the

case of conjunctive queries (with or without arithmetic comparisons), in [2] for disjunctive views, in [48,19,30] for queries with aggregates, in [23] for recursive queries and nonrecursive views, and in [11,12] for several variants of regular path queries. Comprehensive frameworks for view-based query answering, as well as several interesting results for various query languages, are presented in [29,1].

Query answering using views is tightly related to query rewriting [38,23,51]. In particular, [3] studies rewriting of conjunctive queries using conjunctive views whose atoms are DL concepts or roles (the DL used is less expressive thatn $\mathcal{DLR}$). In general, a *rewriting* of a query with respect to a set of views is a function that, given the extensions of the views, returns a set of tuples that is contained in the answer set of the query with respect to the views. Usually, one fixes a priori the language in which to express rewritings (e.g., unions of conjunctive queries), and then looks for the best possible rewriting expressible in such a language. On the other hand, we may call *perfect* a rewriting that returns exactly the answer set of the query with respect to the views, independently of the language in which it is expressed. Hence, if an algorithm for answering queries using views exists, it can be viewed as a perfect rewriting [13,14]. The results presented here show the existence of perfect, and hence maximal, rewritings in a setting where the mediated schema, the views, and the query are expressed in $\mathcal{DLR}$.

6   Conclusions

We have illustrated a logic-based framework for data integration, and in particular for the problem of query answering using views in a data integration system. We have addressed the problem for the case of non-recursive datalog queries posed to a mediated schema expressed in $\mathcal{DLR}$. We have considered different assumptions on the view extensions (sound, complete, and exact), and we have presented a technique that solves the problem in 2EXPTIME worst case computational complexity.

We have seen in the previous section that an algorithm for answering queries using views is in fact a perfect rewriting. For the setting presented here, it remains open to find perfect rewritings expressed in a more declarative query language. Moreover it is of interest to find maximal rewritings belonging to well behaved query languages, in particular, languages with polynomial data complexity, even though we already know that such rewritings cannot be perfect [13].

Acknowledgments

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. Foto N. Afrati, Manolis Gergatsoulis, and Theodoros 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-Verlag, 1999.

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

4. Mokrane Bouzeghoub and Maurizio Lenzerini. Special issue on data extraction, cleaning, and reconciliation. *Information Systems*, 26(8), pages 535–536, 2001.

5. Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Conjunctive query containment in Description Logics with n-ary relations. In *Proc. of the 1997 Description Logic Workshop (DL'97)*, pages 5–9, 1997.

6. Diego Calvanese, Giuseppe De Giacomo, and Maurizio 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.

7. Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Answering queries using views over description logics knowledge bases. In *Proc. of the 17th Nat. Conf. on Artificial Intelligence (AAAI 2000)*, pages 386–391, 2000.

8. Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, and Riccardo 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.

9. Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, and Riccardo Rosati. Information integration: Conceptual modeling and reasoning support. In *Proc. of the 6th Int. Conf. on Cooperative Information Systems (CoopIS'98)*, pages 280–291, 1998.

10. Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, and Riccardo Rosati. Data integration in data warehousing. *Int. J. of Cooperative Information Systems*, 10(3), pages 237–271, 2001.

11. 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.

12. Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Query processing using views for regular path queries with inverse. In *Proc. of the 19th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2000)*, pages 58–66, 2000.

13. 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.

14. Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. What is query rewriting? In *Proc. of the 7th Int. Workshop on Knowledge Representation meets Databases (KRDB 2000)*, pages 17–27. CEUR Electronic Workshop Proceedings, `http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-29/`, 2000.

15. Diego Calvanese, Maurizio Lenzerini, and Daniele Nardi. Description logics for conceptual data modeling. In Jan Chomicki and Günter Saake, editors, *Logics for Databases and Information Systems*, pages 229–264. Kluwer Academic Publisher, 1998.

16. Tiziana Catarci and Maurizio Lenzerini. Representing and using interschema knowledge in cooperative information systems. *J. of Intelligent and Cooperative Information Systems*, 2(4):375–398, 1993.

17. 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.

18. P. P. Chen. The Entity-Relationship model: Toward a unified view of data. *ACM Trans. on Database Systems*, 1(1):9–36, March 1976.

19. Sara Cohen, Werner Nutt, and Alexander 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.

20. Giuseppe De Giacomo and Maurizio Lenzerini. What's in an aggregate: Foundations for description logics with tuples and sets. In *Proc. of the 14th Int. Joint Conf. on Artificial Intelligence (IJCAI'95)*, pages 801–807, 1995.

21. Giuseppe De Giacomo and Maurizio Lenzerini. TBox and ABox reasoning in expressive description logics. In Luigia C. Aiello, John Doyle, and Stuart C. Shapiro, editors, *Proc. of the 5th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'96)*, pages 316–327. Morgan Kaufmann, Los Altos, 1996.

22. Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Andrea Schaerf. $\mathcal{AL}$-log: Integrating Datalog and description logics. *J. of Intelligent Information Systems*, 10(3):227–252, 1998.

23. 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.

24. Ramez A. ElMasri and Shamkant B. Navathe. *Fundamentals of Database Systems*. Benjamin and Cummings Publ. Co., Menlo Park, California, 1988.

25. M. Fattorosi-Barnaba and F. De Caro. Graded modalities I. *Studia Logica*, 44:197–221, 1985.

26. Michael J. Fischer and Richard E. Ladner. Propositional dynamic logic of regular programs. *J. of Computer and System Sciences*, 18:194–211, 1979.

27. Daniela Florescu, Alon Y. Levy, Ioana Manolescu, and Dan 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.

28. Helena Galhardas, Daniela Florescu, Dennis Shasha, and Eric Simon. An extensible framework for data cleaning. Technical Report 3742, INRIA, Rocquencourt, 1999.

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

30. Stéphane Grumbach, Maurizio Rafanelli, and Leonardo 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.

31. Ian Horrocks, Ulrike Sattler, Sergio Tessaris, and Stephan Tobies. Query containment using a DLR ABox. Technical Report LTCS-Report 99-15, RWTH Aachen, 1999.

32. Michael N. Huhns, Nigel Jacobs, Tomasz Ksiezyk, Wei-Min Shen an Munindar P. Singh, and Philip E. Cannata. Integrating enterprise information models in Carnot. In *Proc. of the Int. Conf. on Cooperative Information Systems (CoopIS'93)*, pages 32–42, 1993.

33. R. B. Hull and R. King. Semantic database modelling: Survey, applications and research issues. *ACM Computing Surveys*, 19(3):201–260, September 1987.

34. Matthias Jarke, Maurizio Lenzerini, Yannis Vassiliou, and Panos Vassiliadis, editors. *Fundamentals of Data Warehouses*. Springer-Verlag, 1999.

35. Dexter Kozen and Jerzy Tiuryn. Logics of programs. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science — Formal Models and Semantics*, pages 789–840. Elsevier Science Publishers (North-Holland), Amsterdam, 1990.

36. Maurizio Lenzerini and Andrea Schaerf. Concept languages as query languages. In *Proc. of the 9th Nat. Conf. on Artificial Intelligence (AAAI'91)*, pages 471–476, 1991.

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

38. Alon Y. Levy, Alberto O. Mendelzon, Yehoshua Sagiv, and Divesh 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.

39. Alon Y. Levy and Marie-Christine 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.

40. Alon Y. Levy, Divesh Srivastava, and Thomas Kirk. Data model and query evaluation in global information systems. *J. of Intelligent Information Systems*, 5:121–143, 1995.

41. 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.

42. Chen Li and Edward Chang. On answering queries in the presence of limited access patterns. In *Proc. of the 8th Int. Conf. on Database Theory (ICDT 2001)*, 2001.

43. Chen Li, Ramana Yerneni, Vasilis Vassalos, Hector Garcia-Molina, Yannis Papakonstantinou, Jeffrey D. Ullman, and Murty Valiveti. Capability based mediation in TSIMMIS. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 564–566, 1998.

44. Anand Rajaraman, Yehoshua Sagiv, and Jeffrey 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.

45. Marie-Christine Rousset. Backward reasoning in ABoxes for query answering. In *Proc. of the 1999 Description Logic Workshop (DL'99)*, pages 18–22. CEUR Electronic Workshop Proceedings, `http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-22/`, 1999.

46. Andrea Schaerf. *Query Answering in Concept-Based Knowledge Representation Systems: Algorithms, Complexity, and Semantic Issues*. PhD thesis, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", 1994.

47. Klaus Schild. A correspondence theory for terminological logics: Preliminary report. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI'91)*, pages 466–471, Sydney (Australia), 1991.

48. 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.

49. Stephan Tobies. The complexity of reasoning with cardinality restrictions and nominals in expressive description logics. *J. of Artificial Intelligence Research*, 12:199–217, 2000.
50. 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.
51. Jeffrey D. Ullman. Information integration using logical views. In *Proc. of the 6th Int. Conf. on Database Theory (ICDT'97)*, volume 1186 of *Lecture Notes in Computer Science*, pages 19–40. Springer-Verlag, 1997.
52. Wiebe Van der Hoek and Maarten de Rijke. Counting objects. *J. of Logic and Computation*, 5(3):325–345, 1995.
53. 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.
54. Jennifer Widom. Special issue on materialized views and data warehousing. *IEEE Bulletin on Data Engineering*, 18(2), 1995.
55. Ramana Yerneni, Chen Li, Hector Garcia-Molina, and Jeffrey D. Ullman. Computing capabilities of mediators. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 443–454, 1999.
56. Ramana Yerneni, Chen Li, Jeffrey D. Ullman, and Hector Garcia-Molina. Optimizing large join queries in mediation systems. In *Proc. of the 7th Int. Conf. on Database Theory (ICDT'99)*, pages 348–364, 1999.