Data Integration through *DL-Lite*$_\mathcal{A}$ Ontologies

Diego Calvanese[1], Giuseppe De Giacomo[2], Domenico Lembo[2],
Maurizio Lenzerini[2], Antonella Poggi[2], Riccardo Rosati[2], Marco Ruzzi[2]

[1] Faculty of Computer Science
Free University of Bozen-Bolzano
calvanese@inf.unibz.it

[2] Dip. di Informatica e Sistemistica
SAPIENZA Università di Roma
lastname@dis.uniroma1.it

Abstract. The goal of data integration is to provide a uniform access to a set of heterogeneous data sources, freeing the user from the knowledge about where the data are, how they are stored, and how they can be accessed. One of the outcomes of the research work carried out on data integration in the last years is a clear conceptual architecture, comprising a global schema, the source schema, and the mapping between the source and the global schema. In this paper, we present a comprehensive approach to, and a complete system for, ontology-based data integration. In this system, the global schema is expressed in terms of a TBox of the tractable Description Logics *DL-Lite*$_\mathcal{A}$, the sources are relations, and the mapping language allows for expressing GAV sound mappings between the sources and the global schema. The mapping language has specific mechanisms for addressing the so-called impedance mismatch problem, arising from the fact that, while the data sources store values, the instances of concepts in the ontology are objects. By virtue of the careful design of the various languages used in our system, answering unions of conjunctive queries can be done through a very efficient technique (LOGSPACE with respect to data complexity) which reduces this task to standard SQL query evaluation. We also show that even very slight extensions of the expressive abilities of our system lead beyond this complexity bound.

1 Introduction

The goal of data integration is to provide a uniform access to a set of heterogeneous data sources, freeing the user from the knowledge about where the data are, how they are stored, and how they can be accessed. The problem of designing effective data integration solutions has been addressed by several research and development projects in the last years. Starting from the late 90s, research in data integration has mostly focused on declarative approaches (as opposed to procedural ones) [32, 26]. One of the outcomes of this research work is a clear conceptual architecture for (mediator-based[1]) data integration. According to this architecture [26], the main components of a data integration system are the global schema, the sources, and the mapping. Thus, a data integration system is seen as a triple $\langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, where:

- $\mathcal{G}$ is the *global schema*, providing both a conceptual representation of the application domain, and a reconciled, integrated, and virtual view of the underlying sources.

[1] Other architectures, e.g. [4], are outside the scope of this paper.

– $\mathcal{S}$ is the *source schema*, i.e., the schema of the sources where real data are stored.
– $\mathcal{M}$ is the *mapping* between $\mathcal{G}$ and $\mathcal{S}$, constituted by a set of assertions establishing the connection between the elements of the global schema and those of the source schema. Two basic approaches have been proposed in the literature. The first approach, called *global-as-view* (or simply GAV) [11, 18, 31, 20], focuses on the elements of the global schema, and associates to each of them a view (query) over the sources. On the contrary, in the second approach, called *local-as-view* (or simply LAV) [23, 15, 10], the focus is on the sources, in the sense that a view (query) over the global schema is associated to each of them.

We use the term "data integration management system" to denote a software tool supporting the conceptual architecture described above. Among the various services to be provided by a data integration management system, we concentrate on query answering: Queries are posed in terms of the global schema, and are to be answered by suitably reasoning on the global schema, and exploiting the mappings to access data at the sources.

Data integration is still one of the major challenges in Information Technology. One of the reasons is that large amounts of heterogeneous data are nowadays available within an organization, but these data have been often collected and stored by different applications and systems. Therefore, the need of accessing data by means of flexible and unified mechanisms is becoming more and more important. On the other hand, current commercial data integration tools have several drawbacks. In particular, none of them realizes the goal of describing the global schema independently from the sources. In particular, these tools do not allow for specifying integrity constraints in the global schema, and this implies that the global schema is a sort of data structure for accommodating a reconciled view of the source data, rather than a faithful description of the application domain. It follows that current state-of-the-art data integration tools do not support the conceptual architecture mentioned above.

In this paper, we present a comprehensive approach to, and a complete management system for ontology-based data integration. The system, called MASTRO-I, is based on the following principles:

– The system fully adheres to the conceptual architecture developed by the scientific community.
– The global schema is specified in terms of an ontology, specifically in terms of a TBox expressed in a tractable Description Logics, namely *DL-Lite*$_{\mathcal{A}}$. So, our approach conforms to the view that the global schema of a data integration system can be profitably represented by an ontology, so that clients can rely on a shared conceptualization when accessing the services provided by the system.
– The source schema is the schema of a relational database. In fact, such a schema may result from the federation of a set of heterogeneous, possibly non-relational, data sources. This can be realized by means of a data federation tool, which presents, without materializing them, physical data sources to MASTRO-I as they were a single relational database, obtained by simply transforming each source into a set of virtual relational views and taking their union.
– The mapping language allows for expressing GAV *sound* mappings between the sources and the global schema. A GAV sound mapping specifies that the exten-

sion of a source view provides a subset of the tuples satisfying the corresponding element of the global schema.
Moreover, the mapping language has specific mechanisms for addressing the so-called *impedance mismatch* problem. This problem arises from the fact that, while the data sources store values, the instances of concepts in the ontology (global schema) are objects, each one denoted by an identifier (e.g., a constant in logic), not to be confused with any data item.

MASTRO-I is based on the system QUONTO [1], a reasoner for *DL-Lite*$_{\mathcal{A}}$, and is coupled with a data federation tool that is in charge of federating physical data sources[2].

We point out that our proposal is not the first one advocating the use of ontologies in data integration (see, for example, [12, 2]). However, to the best of our knowledge, MASTRO-I is the first data integration management system addressing simultaneously the following aspects:

- providing a solution to the impedance mismatch problem;
- answering unions of conjunctive queries posed to the global schema according to a method which is sound and complete with respect to the semantics of the ontology;
- careful design of the various languages used in the system, resulting in a very efficient technique (LOGSPACE with respect to data complexity), which reduces query answering to standard SQL query evaluation over the sources.

In order to demonstrate feasibility and efficiency of the MASTRO-I approach to data integration, we also describe in this paper an experimentation carried out over a real-world application scenario. More precisely, we discuss some experiments in which we make use of an ontology benchmark modeling the domain of universities to specify the global schema of the integration system, and connect it, via MASTRO-I mappings, to real data stored at different information systems owned by SAPIENZA University of Rome.

Although in the present work we make use of GAV mappings, the presence of constraints expressed in a rich ontology language in the global schema, makes query answering in our setting more similar to what is carried out in LAV data integration systems rather than in GAV systems. Indeed, while in general GAV systems have been realized as (simple) hierarchies of wrappers and mediators, query answering in LAV can be considered a form of reasoning in the presence of incomplete information, and thus significantly more complex. Early systems based on this approach, like Information Manifold (IM) [28, 29], or INFOMASTER [19, 16], have implemented algorithms [28] for rewriting queries using views, where the views are the ones specified through the (conjunctive) queries in the mappings. The relationship between LAV and GAV data integration systems is explored in [5], where it is indeed shown that a LAV system can be converted into a GAV one by introducing suitable inclusion dependencies in the global schema. If no functionality assertions are present in the global schema, such inclusion

[2] The current implementation of MASTRO-I makes use of Websphere Federation Server, the IBM tool for data federation (`http://www-306.ibm.com/software/data/integration/federation_server/`). However, MASTRO-I can be coupled with any data federation tool based on SQL.

dependencies can then be dealt with in a way similar to what is done here for concept and role inclusions in *DL-Lite*$_\mathcal{A}$. We show in this paper, however, that this is no longer possible in the presence of functionality assertions.

Indeed, one might wonder whether the expressive power of the data integration framework underlying MASTRO-I can be improved. We answer this question by showing that even very slight extensions of the expressive abilities of MASTRO-I in modeling the three components of a data integration system lead beyond the LOGSPACE complexity bound.

We finally point out that MASTRO-I addresses the problem of data integration, and not the one of schema or ontology integration. In other words, MASTRO-I is not concerned with the task of building the ontology representing the global schema starting from the source schema, or from other ontologies. This task, which is strongly related to other important problems, such as database schema integration [3], and ontology alignment, matching, merging, or integration (see, e.g., [17]), are outside the scope of MASTRO-I.

The paper is organized as follows. In Section 2, we describe in detail the various components of the data integration framework adopted in MASTRO-I. In Section 3, we provide a description of the semantics of a data integration system managed by MASTRO-I. In Section 4, we illustrate the basic characteristics of the query answering algorithm. In Section 5, we present our experiments. Finally, in Section 6 we study possible extensions to the MASTRO-I framework, and in Section 7 we conclude the paper.

2 The data integration framework

In this section we instantiate the conceptual architecture for data integration systems introduced in Section 1, by describing the form of the global schema, the source schema, and the mapping for data integration systems managed by MASTRO-I.

2.1 The global schema

Global schemas managed by MASTRO-I are given in terms of TBoxes expressed in *DL-Lite*$_\mathcal{A}$ [30], a DL of the *DL-Lite* family. Besides the use of concepts and roles, denoting sets of objects and binary relations between objects, respectively, *DL-Lite*$_\mathcal{A}$ allows one to use value-domains, a.k.a. concrete domains, denoting unbounded sets of (data) values, and concept attributes, denoting binary relations between objects and values[3]. In particular, the value-domains that we consider here are those corresponding to unbounded (i.e., value-domains with an unbounded size) RDF data types, such as integers, real, strings, etc.

To describe *DL-Lite*$_\mathcal{A}$, we first introduce the DL *DL-Lite*$_{\mathcal{FR}}$, which combines the main features of two DLs presented in [8], called *DL-Lite*$_\mathcal{F}$ and *DL-Lite*$_\mathcal{R}$, respectively. We use the following notation:

[3] In fact, all results presented in [30] and exploited in the present paper can be extended to include role attributes, user-defined domains, as well as inclusion assertions over such domains (see also [7]).

- A denotes an *atomic concept*, B a *basic concept*, C a *general concept*, and $\top_C$ the *universal concept*;
- E denotes a *basic value-domain*, i.e., the range of an attribute, $T_1, \ldots, T_n$ denote the n pairwise disjoint *unbounded RDF data types* used in our logic, and F denotes a *general value-domain*, which can be either an unbounded RDF data type T_i or the *universal value-domain* $\top_D$;
- P denotes an *atomic role*, Q a *basic role*, and R a *general role*;
- U denotes an *atomic attribute*, and V a *general attribute*.

Given an attribute U, we call *domain* of U, denoted by $\delta(U)$, the set of objects that U relates to values, and we call *range* of U, denoted by $\rho(U)$, the set of values related to objects by U.

We are now ready to define *DL-Lite*$_{\mathcal{FR}}$ expressions as follows.

- Basic and general *concept expressions*:

$$B \ ::= \ A \ \mid \ \exists Q \ \mid \ \delta(U) \qquad\qquad C \ ::= \ B \ \mid \ \neg B$$

- Basic and general *value-domain expressions*:

$$E \ ::= \ \rho(U) \qquad\qquad F \ ::= \ \top_D \ \mid \ T_1 \ \mid \ \cdots \ \mid \ T_n$$

- General *attribute expressions*:

$$V \ ::= \ U \ \mid \ \neg U$$

- Basic and general *role expressions*:

$$Q \ ::= \ P \ \mid \ P^- \qquad\qquad R \ ::= \ Q \ \mid \ \neg Q$$

A *DL-Lite*$_{\mathcal{FR}}$ *TBox* allows one to represent intensional knowledge by means of assertions of the following forms:

- *Inclusion assertions*:

$B \sqsubseteq C$	*concept inclusion assertion*
$Q \sqsubseteq R$	*role inclusion assertion*
$E \sqsubseteq F$	*value-domain inclusion assertion*
$U \sqsubseteq V$	*attribute inclusion assertion*

  A concept inclusion assertion expresses that a (basic) concept B is subsumed by a (general) concept C. Analogously for the other types of inclusion assertions.
- *Functionality assertions* on atomic attributes or basic roles:

$$(\mathsf{funct}\ I) \qquad \textit{functionality assertion}$$

  where I denotes either an atomic attribute or a basic role.

DL-Lite$_\mathcal{A}$ TBoxes are *DL-Lite*$_{\mathcal{FR}}$ TBoxes with suitable limitations in the combination of *DL-Lite*$_{\mathcal{FR}}$ TBox assertions. To describe such limitations we first introduce some preliminary notions.

An atomic attribute U (resp., an atomic role P) is called a *functional property in a TBox* $\mathcal{T}$, if $\mathcal{T}$ contains a functionality assertion $(\mathsf{funct}\ U)$ (resp., either $(\mathsf{funct}\ P)$ or $(\mathsf{funct}\ P^-)$). Then, an atomic attribute is called *primitive in* $\mathcal{T}$, if it does not appear positively in the right-hand side of an attribute inclusion assertion of $\mathcal{T}$, i.e., an atomic attribute U_1 is primitive in $\mathcal{T}$ if there does not exist an atomic attribute U_2 such that $U_2 \sqsubseteq U_1$ is asserted in $\mathcal{T}$. Similarly, an atomic role is called *primitive in* $\mathcal{T}$ if it or its inverse do not appear positively in the right-hand side of a role inclusion assertion of $\mathcal{T}$, i.e., an atomic role P is primitive in $\mathcal{T}$ if there does not exist a basic role Q such that neither $Q \sqsubseteq P$ nor $Q \sqsubseteq P^-$ is asserted in $\mathcal{T}$.

Then, *a DL-Lite*$_\mathcal{A}$ *TBox is a DL-Lite*$_{\mathcal{FR}}$ *TBox* $\mathcal{T}$ *satisfying the condition that every functional property in* $\mathcal{T}$ *is primitive in* $\mathcal{T}$.

Roughly speaking, in our logic, *functional properties cannot be specialized*, i.e., they cannot be used positively in the right-hand side of role/attribute inclusion assertions. As shown in [30], reasoning over a *DL-Lite*$_\mathcal{A}$ knowledge base (constituted by a TBox and an ABox, which specifies the instances of concept and roles) is tractable. More precisely, TBox reasoning is in PTIME and query answering is in LOGSPACE w.r.t. data complexity, i.e., the complexity measured in the size of the ABox only (whereas query answering for *DL-Lite*$_{\mathcal{FR}}$ is PTIME-hard [8]). Thus, *DL-Lite*$_\mathcal{A}$ presents the same computational behavior of all DLs of the *DL-Lite* family, and therefore is particularly suited for integration of large amounts of data.

2.2 The source schema

The source schema in MASTRO-I is assumed to be a flat relational database schema, representing the schemas of all the data sources. Actually, this is not a limitation of the system, since the source schema coupled with MASTRO-I can be the schema managed by a data federation tool. So, the data federation tool is in charge of interacting with data sources, presenting them to MASTRO-I as a single relational database schema, obtained by wrapping physical sources, possibly heterogeneous, and not necessarily in relational format. Furthermore, the data federation tool is in charge of answering queries formulated over the source schema, by suitably transforming and asking them to the right sources, finally combining the single results into the overall answer. In other words, the data federation tool makes MASTRO-I independent from the physical nature of the sources, by providing a logical representation of them (physical independence), whereas MASTRO-I is in turn in charge of making all logical aspects transparent to the user, by maintaining the conceptual global schema separate from the logical federated schema, and connecting them via suitable mappings (logical independence).

2.3 The mapping

The mapping in MASTRO-I establishes the relationship between the source schema and the global schema, thus specifying how data stored at the sources are linked to the instances of the concepts and the roles in the global schema. To this aim, the mapping

specification takes suitably into account the impedance mismatch problem, i.e., the mismatch between the way in which data is (and can be) represented in a data source, and the way in which the corresponding information is rendered through the global schema.

The MASTRO-I mapping assertions keep data value constants separate from object identifiers, and construct identifiers as (logic) terms over data values. More precisely, object identifiers in MASTRO-I are *terms* of the form $f(d_1, \ldots, d_n)$, called object terms, where f is a function symbol of arity $n > 0$, and $d_1, \ldots, d_n$ are data values stored at the sources. Note that this idea traces back to the work done in deductive object-oriented databases [22].

We detail below the above ideas. The mapping in MASTRO-I is specified through a set of mapping assertions, each of the form

$$\Phi(\boldsymbol{v}) \rightsquigarrow S(\boldsymbol{w})$$

where

- $\Phi(\boldsymbol{v})$, called the *body* of the mapping, is a first-order logic (FOL) query of arity $n > 0$, with distinguished variables $\boldsymbol{v}$, over the source schema $\mathcal{S}$ (we will write such query in the SQL syntax), and
- $P(\boldsymbol{w})$, called the *head*, is an atom where S can be an atomic concept, an atomic role, or an atomic attribute occurring in the global schema $\mathcal{G}$, and $\boldsymbol{w}$ is a sequence of terms.

We define three different types of mapping assertions:

1. *Concept mapping assertions*, in which the head is a unary atom of the form $A(f(\boldsymbol{v}))$, where A is an atomic concept and f is a function symbol of arity n;
2. *Role mapping assertions*, in which the head is a binary atom of the form $P(f_1(\boldsymbol{v}'), f_2(\boldsymbol{v}''))$, where P is an atomic role, f_1 and f_2 are function symbols of arity $n_1, n_2 > 0$, and $\boldsymbol{v}'$ and $\boldsymbol{v}''$ are sequences of variables appearing in $\boldsymbol{v}$;
3. *Attribute mapping assertions*, in which the head is a binary atom of the form $U(f(\boldsymbol{v}'), v'' : T_i)$, where U is an atomic attribute, f is a function symbol of arity $n' > 0$, $\boldsymbol{v}'$ is a sequence of variables appearing in $\boldsymbol{v}$, v'' is a variable appearing in $\boldsymbol{v}$, and T_i is an RDF data type.

In words, such mapping assertions are used to map source relations (and the tuples they store), to concepts, roles, and attributes of the ontology (and the objects and the values that constitute their instances), respectively. Note that an attribute mapping also specifies the type of values retrieved by the source database.

Example 1. Consider the following mapping assertions:

M_1 : `SELECT S_CODE FROM STUDENTS WHERE DOB <= '1990/01/01'` $\rightsquigarrow$ *Student*(**st**(`S_CODE`))

M_2 : `SELECT S_CODE, S_NAME FROM STUDENTS WHERE DOB <= '1990/01/01'` $\rightsquigarrow$ *name*(**st**(`S_CODE`), `S_NAME` : `xsd:string`)

M_3 : `SELECT S_CODE, C_CODE FROM COURSES` $\rightsquigarrow$ *takesCourse*(**st**(`S_CODE`), **co**(`C_CODE`))

Such assertions relate a source schema containing the relations STUDENTS and COURSES to a global schema containing the atomic concept *Student*, the atomic attribute *name*, and the atomic role *takesCourse*. M_1 is a concept mapping assertion that selects from the table STUDENTS the code (variable S_CODE) of students whose date of birth (variable DOB) is after December 31, 1989, and for each such code builds, by means of the function symbol **st**, an object term representing an instance of the concept *Student*. M_2 is an attribute mapping assertion that, besides the codes, selects also the names (variable S_NAME) of the students born after December 31, 1989, and for each selected tuple builds a pair constituted by a term of the form **st**(S_CODE), which is as in assertion M_1, and a constant representing the name of the student. To such a constant M_2 assigns the data type xsd:string. Finally, M_3 is a role mapping assertion relating the relation COURSES to the global atomic role *takesCourse*. More precisely, from each pair constituted by the code of a student (variable S_CODE) and the code of a course she takes (variable C_CODE), M_3 builds a pair of object terms (**st**(S_CODE),**co**(C_CODE)), where **co** is a function symbol used to build object terms representing courses taken by students.

We point out that, in fact, the body of each mapping assertion is never really evaluated in order to extract values from the sources to build instances of the global schema, but rather it is used to unfold queries posed over the global schema, and thus rewriting them into queries posed over the source schema (cf. Section 4). Also, we notice that the mapping designer has to specify a correct *DL-Lite*$_\mathcal{A}$ data type for the values extracted from the source, in order to guarantee coherency of the system. This aspect is detailed in the next section.

3 Semantics

We now illustrate the semantics of a data integration system managed by MASTRO-I.

Let $\mathcal{J} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a data integration system. The general idea is to start with a database D for the source schema $\mathcal{S}$, i.e., an extension of the data sources, called the *source database* for $\mathcal{J}$. The source database D has to be coherent with the typing assertions that implicitly appears in the mapping $\mathcal{M}$. More precisely, this means that, for every attribute mapping $\Phi(\boldsymbol{v}) \rightsquigarrow U(f(\boldsymbol{v'}), v'' : T_i)$, the values for v'' extracted from D are of type T_i. In the rest of this paper, we always assume that the source database is coherent with the mapping.

Given a source database D, we define the semantics of $\mathcal{J}$ as the set of interpretations for $\mathcal{G}$ that both satisfy the TBox assertions of $\mathcal{G}$, and satisfy the mapping assertions in $\mathcal{M}$ with respect to D. This informal definition makes use of different notions that we detail below.

- First, the notion of interpretation for $\mathcal{G}$ is the usual one in DL. An *interpretation* $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ for $\mathcal{G}$ consists of an interpretation domain $\Delta^\mathcal{I}$ and an *interpretation function* $\cdot^\mathcal{I}$. $\Delta^\mathcal{I}$ is the disjoint union of the domain of objects $\Delta^\mathcal{I}_O$, and the domain of values $\Delta^\mathcal{I}_V$, while the interpretation function $\cdot^\mathcal{I}$ assigns the standard formal meaning to all expressions and assertions of the logic *DL-Lite*$_\mathcal{A}$ (see [7]). The only aspect which is not standard here is the need of dealing with objects denoted by terms (see

previous section). To this end, we now introduce two disjoint alphabets, called Γ_O and Γ_V, respectively. Symbols in Γ_O are called object terms, and are used to denote objects, while symbols in Γ_V, called value constants, are used to denote data values. More precisely, Γ_O is built starting from Γ_V and a set Λ of function symbols of any arity (possibly 0), as follows: If $f \in \Lambda$, the arity of f is n, and $d_1, \ldots, d_n \in \Gamma_V$, then $f(d_1, \ldots, d_n)$ is a term in Γ_O, called *object term*. In other words, object terms are either functional terms of arity 0, called object constants, or terms constituted by a function symbol applied to data value constants. We are ready to state how the interpretation function $\cdot^{\mathcal{I}}$ treats Γ_V and Γ_O: $\cdot^{\mathcal{I}}$ assigns a different value in $\Delta_V^{\mathcal{I}}$ to each symbol in Γ_V, and a different element of $\Delta_O^{\mathcal{I}}$ to every object term (not only object constant) in Γ_O. In other words, *DL-Lite*$_{\mathcal{A}}$ enforces the unique name assumption on both value constants and object terms.

– To the aim of describing the semantics of mapping assertions with respect to a database D for the source schema $\mathcal{S}$, we first assume that all data values stored in the database D belong to Γ_V[4]. Then, if q is a query over the source schema $\mathcal{S}$, we denote by $ans(q, D)$ the set of tuples obtained by evaluating the query q over the database D (if q has no distinguished variables, then $ans(q, D)$ is a boolean). Finally, we introduce the notion of ground instance of a formula. Let γ be a FOL formula with free variables $\boldsymbol{x} = (x_1, \ldots, x_n)$, and let $\boldsymbol{s} = (s_1, \ldots, s_n)$ be a tuple of elements in $\Gamma_V \cup \Gamma_O$. A ground instance $\gamma[\boldsymbol{x}/\boldsymbol{s}]$ of γ is obtained from γ by substituting every occurrence of x_i with s_i.
We are now ready to specify the semantics of mapping assertions. We say that an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ *satisfies* the mapping assertion $\Phi(\boldsymbol{v}) \rightsquigarrow S(\boldsymbol{w})$ with respect to D, if for every ground instance

$$\Phi[\boldsymbol{v}/\boldsymbol{s}] \rightsquigarrow S[\boldsymbol{v}/\boldsymbol{s}]$$

of $\Phi(\boldsymbol{v}) \rightsquigarrow S(\boldsymbol{w})$, we have that $ans(\Phi[\boldsymbol{v}/\boldsymbol{s}], D) = true$ implies $S[\boldsymbol{v}/\boldsymbol{s}]^{\mathcal{I}} = true$, where, for a ground atom $S(\boldsymbol{t})$, with $\boldsymbol{t} = (t_1, \ldots, t_n)$ a tuple of terms, we have that $S(\boldsymbol{t})^{\mathcal{I}} = true$ if and only if $(t_1^{\mathcal{I}}, \ldots, t_n^{\mathcal{I}}) \in p^{\mathcal{I}}$. Note that the above definition formalizes the notion of sound mapping, as it treats each mapping assertion as an implication.
– With the above notion in place, we define the semantics of $\mathcal{J}$ with respect to D as follows:

$$sem_D(\mathcal{J}) = \{ \text{ interpretation } \mathcal{I} \mid \mathcal{I} \text{ satisfies all assertions in } \mathcal{G} \text{ and } \mathcal{M} \text{ wrt } D \}$$

We say that $\mathcal{J}$ is satisfiable withe respect the D if $sem_D(\mathcal{J}) \neq \emptyset$.

Among the various reasoning services that can be considered over a data integration system, in this paper we are interested in the problem of answering unions of conjunctive queries (UCQs) posed to the global schema. The semantics of query answering is given in terms of certain answers to the query, defined as follows. Given a data integration system $\mathcal{J} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, and a database D for $\mathcal{S}$, the set of *certain answers* to the query $q(\boldsymbol{x})$ over $\mathcal{G}$ with respect to D (denoted by $cert(q, \mathcal{J}, D)$) is the set of all tuples $\boldsymbol{t}$ of elements of $\Gamma_V \cup \Gamma_O$ such that $q[\boldsymbol{x}/\boldsymbol{t}]$ is true in every $\mathcal{I} \in sem_D(\mathcal{J})$.

[4] We could also introduce suitable conversion functions in order to translate values stored at the sources into value constants in Γ_V, but we do not deal with this issue here.

4 Query answering

In this section, we sketch our query answering technique (more details can be found in [30]). Consider a data integration system $\mathcal{J} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ and a database D for $\mathcal{S}$, and assume that $\mathcal{J}$ is satisfiable with respect to D, i.e., $sem_D(\mathcal{J}) \neq \emptyset$.

We start with the following observation. Suppose we evaluate (over D) the queries in the left-hand sides of the mapping assertions, and we materialize accordingly the corresponding assertions in the right-hand sides. This would lead to a set of ground assertions, that can be considered as a *DL-Lite* ABox[5], denoted by $\mathcal{A}^{\mathcal{M},D}$. It can be shown that query answering over $\mathcal{J}$ and D can be reduced to query answering over the *DL-Lite*$_\mathcal{A}$ knowledge base constituted by the TBox $\mathcal{G}$ and the ABox $\mathcal{A}^{\mathcal{M},D}$. However, due to the materializion of $\mathcal{A}^{\mathcal{M},D}$, the query answering algorithm resulting from this approach would be polynomial in the size of D. On the contrary, our idea is to avoid any ABox materialization, but rather answer Q by reformulating it into a new query that can be afterwards evaluated directly over the database D. The resulting query answering algorithm is constituted by four steps, which are called *rewriting*, *filtering*, *unfolding* and *evaluation*, and are described in the following.

4.1 Rewriting

Given a UCQ Q over a data integration system $\mathcal{J} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, and a source database D for $\mathcal{J}$, the rewriting step computes a new UCQ Q' over $\mathcal{J}$, where the assertions of $\mathcal{G}$ are compiled in. In computing the rewriting, only inclusion assertions of the form $B_1 \sqsubseteq B_2$, $Q_1 \sqsubseteq Q_2$, and $U_1 \sqsubseteq U_2$ are taken into account, where B_i, Q_i, and U_i, with $i \in \{1, 2\}$, are a basic concept, a basic role and an atomic attribute, respectively. Intuitively, the query Q is rewritten according to the knowledge specified in $\mathcal{G}$ that is relevant for answering Q, in such a way that the rewritten query Q' is such that $cert(Q', \langle \emptyset, \mathcal{S}, \mathcal{M} \rangle, D) = cert(Q, \mathcal{J}, D)$, i.e., the rewriting allows to get rid of $\mathcal{G}$.

Example 2. Consider a data integration system $\mathcal{J} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ where $\mathcal{G}$ is *DL-Lite*$_\mathcal{A}$ TBox comprising the concept inclusion assertion *Student* $\sqsubseteq \exists$*takesCourse*, and consider the query $Q(x) :-$ *takesCourse*(x, y) (written in Datalog syntax), which is asking for individuals that take (at least) a course. The output of the rewriting step is the following UCQ Q' (written in Datalog syntax):

$$Q'(x) :- \textit{takesCourse}(x, y).$$
$$Q'(x) :- \textit{Student}(x).$$

Since the global schema says that each student takes at least a course, the query Q' asks both for individuals that take a course and for individuals that are students. Then, we can answer Q' over $\mathcal{J}$ by disregarding the inclusion assertions in $\mathcal{G}$, and we get the same result we got by answering Q over $\mathcal{J}$. In other words, whatever is the underlying source database D, we have that $cert(Q', \langle \emptyset, \mathcal{S}, \mathcal{M} \rangle, D) = cert(Q, \mathcal{J}, D)$.

[5] In DL jargon, an ABox is a set of membership assertions, i.e., assertions stating which are the instances of the concepts and the roles defined in the corresponding TBox.

We refer the reader to [30, 9] for a formal description of the query rewriting algorithm used by MASTRO-I and for a proof of its soundness and completeness. We only notice here that the rewriting procedure does not depend on the source database D, runs in polynomial time in the size of $\mathcal{G}$, and returns a query Q' whose size is at most exponential in the size of Q.

4.2 Filtering

Let Q' be the UCQ produced by the rewriting step above. In the filtering step we take care of a particular problem that the disjuncts, i.e., conjunctive queries, in Q' might have. Specifically, a conjunctive query cq is called *ill-typed* if it has at least one join variable x appearing in two incompatible positions in cq, i.e., such that the TBox $\mathcal{G}$ of our data integration system logically implies that x is both of type T_i, and of type T_j, with $T_i \neq T_j$ (remember that in *DL-Lite*$_\mathcal{A}$ data types are pairwise disjoint). The purpose of the filtering step is to remove from the UCQ Q' all the ill-typed conjunctive queries. Intuitively, such a step is needed because the query Q' has to be unfolded and then evaluated over the source database D (cf. the next two steps of the MASTRO-I query answering algorithm described below). These last two steps, performed for an ill-typed conjunctive query might produce incorrect results. Indeed, the set of certain answers over a satisfiable data integration system for an ill-typed conjunctive query cq is always empty (cf. Section 3). However, the SQL query that results after the unfolding step (see below) is sent to the underlying data federation tool that uses its SQL engine for evaluating it over the source database, and it might happen that the data value conversions carried out by the SQL engine make the evaluation of such an SQL query non-empty, thus incorrectly producing a non-empty set of certain answers for cq. Obviously, this might produce an incorrect result in all those cases in which cq occurs in the UCQ Q'. The filtering step, by simply dropping all ill-typed queries from Q', solves this problem and produces a new UCQ Q'' over $\mathcal{J}$.

Example 3. Consider the boolean conjunctive query

$$Q :- \mathit{id}(x, z), \mathit{age}(x, z).$$

where *id* and *age* are two atomic attributes, asking if there exists a value constant which is both the *id* and the *age* of a certain individual (e.g., a student). Consider now a data integration system $\mathcal{J} = \langle \mathcal{G}, \mathcal{M}, \mathcal{S} \rangle$ where $\mathcal{G}$ contains the assertions

$$\begin{aligned} \rho(\mathit{id}) &\sqsubseteq \texttt{xsd:string} \\ \rho(\mathit{age}) &\sqsubseteq \texttt{xsd:integer} \end{aligned}$$

specifying that the range of *id* is `xsd:string`, and the range of *age* is `xsd:integer`. Obviously, the query Q above is ill-typed, since `xsd:string` and `xsd:integer` are two disjoint data types.

Let us now answer Q over $\mathcal{J}$, and assume that the output of the rewriting step is Q itself (i.e., the global schema does not contain assertions that cause the rewriting of Q). If we skipped now the filtering step, the query above would be handed to the unfolder (see below), which would transform it into an SQL query (according to $\mathcal{M}$).

Then, its evaluation over some source database D might return some tuples, due to some data value conversions that make the join specified in Q succeed. In other words, $cert(Q, \mathcal{J}, D)$ might be non-empty and the query might incorrectly be considered true. The filtering step, instead, simply drops the ill-typed queries from the UCQ to be sent to the unfolder, which in this example is therefore an empty query. As a consequence, $cert(Q, \mathcal{J}, D)$ is correctly empty.

4.3 Unfolding

Given the UCQ Q'' over $\mathcal{J}$ computed by the filtering step, the unfolding step computes, by using logic programming technology, an SQL query Q''' over the source schema $\mathcal{S}$, that possibly returns object terms. It can be shown [30] that Q''' is such that $ans(Q''', D) = cert(Q'', \langle \emptyset, \mathcal{S}, \mathcal{M} \rangle, D)$, i.e., unfolding allows us to get rid of $\mathcal{M}$. Moreover, the unfolding procedure does not depend on D, runs in polynomial time in the size of $\mathcal{M}$, and returns a query whose size is polynomial in the size of Q''.

Example 4. Let us now continue Example 2, and assume that $\mathcal{M}$ is constituted by the mapping assertions described in Example 1. It is easy to see that Q' does not contain ill-typed queries, and therefore the outut of the filtering step is Q' itself. Also, it is easy to see that Q' has to be unfolded by means of mapping assertions M_1 (used to unfold $Q(x) :-$ *takesCourse*(x, y)) and M_3 (used to unfold $Q(x) :-$ *Student*(x)). The SQL query that results from this unfolding is the following:

```
SELECT CONCAT('st(',S_CODE,')') FROM COURSES
UNION
SELECT CONCAT('st(',S_CODE,')') FROM STUDENTS
WHERE DOB <= '1990/01/01'
```

Notice that in the above query we have made use of the SQL function `CONCAT`[6]. Such a function allows us to concatenate strings to construct object terms of the form **st**(`S_CODE`). By virtue of this mechanism, the evaluation of the above query over the source database returns indeed a set of object terms representing individuals (in fact, students), coherently to what the original query Q asks for.

4.4 Evaluation

The evaluation step consists in simply delegating the evaluation of the SQL query Q''', produced by the unfolding step, to the data federation tool managing the data sources. Formally, such a tool returns the set $ans(Q''', D)$, i.e., the set of tuples obtained from the evaluation of Q''' over D.

[6] For simplicity we assume that the underlying data federation tool allows for using `CONCAT` with an arbitrary number of arguments.

4.5 Correctness of query answering

It can be shown that the query answering procedure described above correctly computes the certain answers to UCQs. Based on the computational properties of such an algorithm, we can then characterize the complexity of our query answering method.

Theorem 1. *Let $\mathcal{J} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a* MASTRO-I *data integration system, and D a source database for $\mathcal{J}$. Answering a UCQ over $\mathcal{J}$ with respect to D can be reduced to the evaluation of an SQL query over D, and is* LOGSPACE *in the size of D.*

Finally, we remark that, as we said at the beginning of this section, we have assumed that the data integration system $\mathcal{J}$ is consistent with respect to the database D, i.e., $sem_D(\mathcal{J})$ is non-empty. Notably, it can be shown that all the machinery we have devised for query answering can also be used for checking consistency of $\mathcal{J}$ with respect to D. Therefore, checking consistency can also be reduced to sending appropriate SQL queries to the source database [30].

5 Experimentation

In this section, we comment on the results of an experimentation that we have carried out on a real-world data integration scenario. The main aim of the experimentation is to test on a case of real and practical interest the MASTRO-I architecture for data integration, which reflects the fundamental principle of maintaining separate the physical level of the data sources, which remain autonomous and independent, from the conceptual representation of the domain of discourse, whose design reflects only the ambit of interest and is in principle independent from the specific data at the sources. To this aim, we have considered a set of information sources used by different administrative offices of SAPIENZA University of Rome, and we have used the so-called Lehigh University Benchmark (LUBM)[7] to specify the global schema of our integration system. LUBM consists of an ontology for modeling universities, and it is a de facto standard for benchmarking ontology reasoners. Usually, extensional data for the LUBM intensional level are synthesized in an automatic way, possibly using benchmark generators available for LUBM. Here, instead, by virtue of the mapping mechanism provided by MASTRO-I, we are able to connect our specific data sources to the LUBM ontology, which has been of course designed independently from such data.

5.1 Scenario

As data sources, we consider three legacy relational databases, containing the overall number of 25 relations, each storing from a few tuples up to 50,000 tuples. We make use of IBM Websphere Federation Server to federate the above data sources, in such a way that MASTRO-I can see such sources as a single relational schema (source schema). As already said, to model the global schema we make use of the LUBM ontology. The ontology contains concepts for persons, students, professors, publications, courses, etc.,

[7] `http://swat.cse.lehigh.edu/projects/lubm/`

as well as appropriate relationships for such a universe of discourse. The ontology is specified in OWL-DL[8], and it currently defines 43 classes and 32 properties (including 25 object properties, i.e., roles, and 7 datatype properties, i.e., attributes). Note that, in order to use the LUBM ontology in MASTRO-I, we rephrased it in *DL-Lite*$_\mathcal{A}$, essentially capturing all its constructs[9], and also enriched it with further TBox assertions modeling peculiar aspects of the domain not present in the original ontology. For example, we also considered the role $hasExam$, to model the courses for which a student has passed the exam.

Due to space limits, we provide below only a fragment of the *DL-Lite*$_\mathcal{A}$ ontology used in our experiments.

$$\begin{array}{rcl@{\qquad}rcl} \textit{Student} & \sqsubseteq & \exists \textit{takesCourse} & \exists \textit{takesCourse}^- & \sqsubseteq & \textit{Course} \\ \textit{Course} & \sqsubseteq & \exists \textit{teacherOf}^- & \exists \textit{teacherOf} & \sqsubseteq & \textit{Faculty} \\ \textit{Faculty} & \sqsubseteq & \exists \textit{worksFor} & \exists \textit{worksFor}^- & \sqsubseteq & \textit{University} \\ \textit{University} & \sqsubseteq & \exists \textit{hasAlumnus} & \exists \textit{hasAlumnus}^- & \sqsubseteq & \textit{Student} \end{array}$$

In words, the assertions in the first column, from top to bottom, respectively say that each student must take a course (i.e., must be connected by the role *takesCourse* to a certain individual), each course is necessarily taught by some individual, each faculty participates to the role *worksFor*, each university has at least one alumnus. Assertions in the second column, from top to bottom, respectively say that individuals in the inverse of the role *takesCourse* (resp. the role *teacherOf*, the inverse of the role *worksForUniv*, and the inverse of the role *hasAlumnus*) must be courses (resp. faculties, universities, students).

5.2 Testing query answering

To show scalability of query answering in MASTRO-I, we tuned our system in such a way that data stored at the sources resulted into six different source databases of growing size. Table 1 briefly says how data coming from the information systems of SAPIENZA University of Rome have been filtered for our experiments.

We then considered five significant queries over the global schema, and measured the behavior of MASTRO-I in terms of both the size of the resulting answer sets (i.e., the number of tuples in the answer to each query), and the overall time that MASTRO-I took to produce these answer sets. Below we describe each test query (given in DATALOG notation).

– *Query 1*: It asks for all persons living in Rome

$$Q_1(x) :- \textit{Person}(x), \textit{address}(x, \texttt{'ROMA'}).$$

– *Query 2*: It asks for the names of all students that take a course, together with the name of such a course:

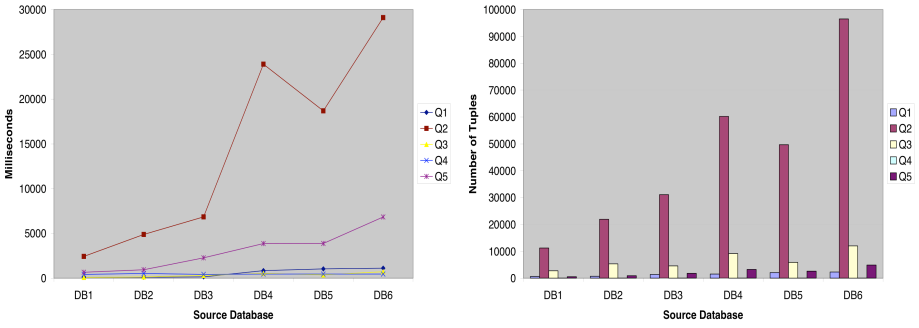
$$Q_2(z, w) :- \textit{Student}(x), \textit{name}(x, z), \textit{takesCourse}(x, y), \textit{name}(y, w).$$

[8] http://www.w3.org/2007/OWL/wiki/OWL_Working_Group

[9] We recall that, since *DL-Lite*$_\mathcal{A}$ is less expressive than OWL-DL, an OWL-DL ontology cannot be in general exactly specified in *DL-Lite*$_\mathcal{A}$.

Name	DB size (number of tuples)	Data description
DB_1	118075	from 1993 to 1995 (restricted to students living in Rome)
DB_2	165049	from 1993 to 1995
DB_3	202305	from 1993 to 1997 (restricted to students living in Rome)
DB_4	280578	from 1993 to 1997
DB_5	328256	from 1993 to 1999 (restricted to students living in Rome)
DB_6	482043	from 1993 to 1999

Table 1. Source databases used for tests



(a) (b)

Fig. 1. Query answering: (a) Execution time (b) Number of tuples in the answer

– *Query 3*: It asks for all persons that passed at least an exam:

$$Q_3(x) :- \mathit{Person}(x), \mathit{hasExam}(x, y).$$

– *Query 4*: It asks for the names of all persons whose address is the same as the address of the organization for which their advisor works:

$$Q_4(z) :- \mathit{Person}(y), \mathit{name}(y, z), \mathit{address}(y, w), \mathit{advisor}(y, x), \mathit{worksFor}(x, v), \mathit{address}(v, w).$$

– *Query 5*: It asks for all students that take a course, together with the address of the organization for which the course teacher works:

$$Q_5(x, c) :- \mathit{Student}(x), \mathit{takesCourse}(x, y), \mathit{teacherOf}(z, y), \mathit{worksFor}(z, w), \mathit{address}(w, c).$$

The results of our experiments are given in Figure 1, which shows the performance (execution time) for answering each query w.r.t. the growth of the size of the source database (Figure 1(a)), and the number of tuples returned by each query for each source database (Figure 1(b)).

All experiments have been carried out on an Intel Pentium IV Dual Core machine, with 3 GHz processor clock frequency, equipped with 1 Gb of RAM, under the operating system Windows XP professional.

5.3 Discussion

For each query, the execution time comprises the time needed for rewriting and filtering the input query (both rewriting and filtering are executed by the system QUONTO, which is at the core of MASTRO-I) unfolding the resulting query, and evaluating the unfolded query over the underlying database (both unfolding and evaluation are delegated to IBM Websphere Federation Server). We point out that the time needed for query rewriting and query unfolding is negligible w.r.t. the overall execution time, and that the major time consuming process is the evaluation of the rewritten and unfolded query over the source database. This depends both on the number of disjuncts occurring in the rewritten query (which is a union of conjunctive queries), and the number of source relations mapped to concepts, roles, and attributes occurring as predicates of the query atoms. As an example, we provide below the rewriting of Query 2 (expressed in Datalog notation), for which we measured the worst performance in terms of execution times (n_0 below denotes a fresh existentially quantified variable introduced by the rewriting process).

$$
\begin{array}{lcl}
Q_2(z, w) & :- & \textit{name}(y, w), \textit{examRating}(x, y, n_0), \textit{name}(x, z). \\
Q_2(z, z) & :- & \textit{takesGraduateCourse}(x, x), \textit{name}(x, z). \\
Q_2(z, w) & :- & \textit{name}(y, w), \textit{takesGraduateCourse}(x, y), \textit{name}(x, z). \\
Q_2(z, w) & :- & \textit{name}(y, w), \textit{hasExam}(x, y), \textit{name}(x, z). \\
Q_2(z, z) & :- & \textit{examRating}(x, x, n_0), \textit{name}(x, z). \\
Q_2(z, z) & :- & \textit{takesCourse}(x, x), \textit{name}(x, z). \\
Q_2(z, z) & :- & \textit{hasExam}(x, x), \textit{name}(x, z). \\
Q_2(z, w) & :- & \textit{name}(y, w), \textit{takesCourse}(x, y), \textit{name}(x, z).
\end{array}
$$

We notice that MASTRO-I shows good scalability w.r.t. the growth of the size of the ABox, and that execution time is always limited, even for answering queries that are rewritten into unions of conjunctive queries with several disjuncts.

6 Extending the data integration framework

In this section we study whether the data integration setting presented above can be extended while keeping the same complexity of query answering. In particular, we investigate possible extensions for all the three components $\langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ of the system.

6.1 Extensions to *DL-Lite*$_\mathcal{A}$

With regard to the logic used to express the global schema $\mathcal{G}$, the results in [8] already imply that it is not possible to go beyond *DL-Lite*$_\mathcal{A}$ (at least by means of the usual DL constructs) and at the same time keep the data complexity of query answering within LOGSPACE. Here we consider the possibility of removing the *unique name assumption* (UNA), i.e., the assumption that, in every interpretation of a data integration system, both two distinct value constants, and two distinct object terms denote two different domain elements. Unfortunately, this leads query answering out of LOGSPACE, as shown by the following theorem.

Theorem 2. *Let $\mathcal{J} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a* MASTRO-I *data integration system extended by removing the UNA, and D a database for $\mathcal{S}$. Answering a UCQ (in fact, a query constituted by a single atom) over $\mathcal{J}$ with respect to D is* NLOGSPACE*-hard in the size of D.*

Proof. We prove the result already for a data integration system in which the global schema is expressed in *DL-Lite*$_{\mathcal{F}}$ [8], a sub-language of *DL-Lite*$_{\mathcal{A}}$, and in which the mapping assertions have the simplest possible form, i.e., they map a single source relation to a single concept or role of the global schema[10]. The proof is based on a reduction from reachability in directed graphs, which is NLOGSPACE-hard.

Let $G = \langle V, E \rangle$ be a directed graph, where V is the set of vertexes and E the set of directed edges. Reachability is the problem of deciding, given two vertexes $s, t \in V$ whether there is an oriented path formed by edges in E in the graph that, starting from s allows to reach t. We consider the graph represented through first-child and next-sibling functional relations F, N, S (cf. Figure 2).

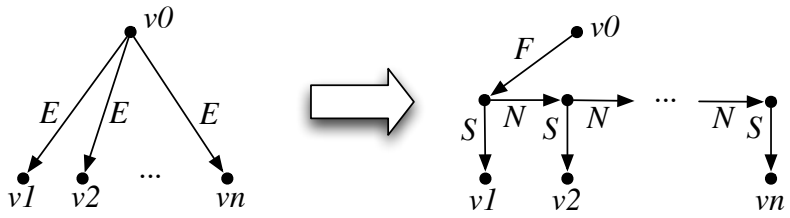


Fig. 2. Representation of a graph through the functional relations F, N, S

We define the data integration system $\mathcal{J}_{una} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ as follows:

- The alphabet of $\mathcal{G}$ consists of the atomic concept A_g and the atomic roles F_g, N_g, S_g, and P_g. $\mathcal{G}$ consists only of the functionality assertions $\{(\mathsf{funct}\ \mathcal{R}_g) \mid \mathcal{R}_g \in \{F_g, N_g, S_g, P_g\}\}$.
- $\mathcal{S}$ contains the binary relational tables F_s, N_s, S_s, and P_s, with columns c_1 and c_2, and the unary relational table A_s, with column c.
- The mapping $\mathcal{M}$ maps each table $\mathcal{R}_s$ to the corresponding role or concept $\mathcal{R}_g$, i.e.,

  $$\texttt{SELECT } c_1, c_2 \texttt{ FROM } \mathcal{R}_s \rightsquigarrow \mathcal{R}_g(\mathbf{id}(c_1), \mathbf{id}(c_2)), \quad \text{for } \mathcal{R} \in \{F, N, S, P\}$$
  $$\texttt{SELECT } c \texttt{ FROM } A_s \rightsquigarrow A_g(\mathbf{id}(c))$$

  Notice that we are using a single function symbol **id** (that we intend to represent the identity function).

Then, from the graph G and the two vertexes s, t, we define the source database D_G as follows:

$$D_G = \{\mathcal{R}_s(a, b), \mathcal{R}_s(a', b') \mid (a, b) \in \mathcal{R}, \text{ for } \mathcal{R} \in \{F, N, S\}\} \cup \{P_s(init, s), P_s(init, s')\} \cup \{A_s(t)\}$$

[10] The mapping assertions actually play no role in the proof, and the hardness result holds already for a plain *DL-Lite*$_{\mathcal{F}}$ knowledge base constituted by a TBox and an ABox.

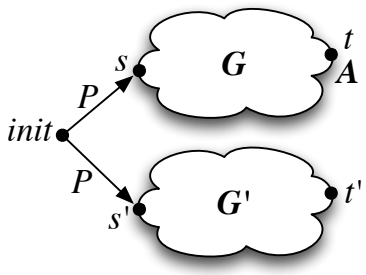


Fig. 3. Structure of the database used in the proof of Theorem 2

In other words, we encode in D_G two copies of (the representation of) the graph G. In addition, we include in D_G the facts $A_s(t)$, $P_s(init, s)$, and $P_s(init, s')$, where $init$ is a database constant that does not correspond to any vertex of (the representation of) G (cf. Figure 3).

It is now possible to prove that t is reachable from s in G iff $\mathbf{id}(t') \in cert(Q, \mathcal{J}_{una}, D_G)$, where $Q(x) :- A_g(x)$ is the query returning the instances of A_g. Indeed, it is easy to verify that the latter holds if and only if $\mathbf{id}(t)$ and $\mathbf{id}(t')$ are the same object in every interpretation in $sem_{D_G}(\mathcal{J}_{una})$, i.e., the equality $\mathbf{id}(t) = \mathbf{id}(t')$ is entailed by $\mathcal{J}_{una}$. This is the case if and only if $\mathbf{id}(t)$ and $\mathbf{id}(t')$ are forced to be equal by the functionality of the roles P_g, F_g, N_g, and S_g. Given the structure of the database D_G, such an equality is enforced if and only if t is reachable from s in G. □

Notice that a simple variation of the above proof can be used to show that query answering, and in particular instance checking already, in *DL-Lite*$_{\mathcal{F}}$ without the unique name assumption is NLOGSPACE-hard with respect to data complexity.

6.2 Different source schemas

Although MASTRO-I is currently only able to deal with relational sources, managed by a relational data federation tool, it is not hard to see that all the results presented in this paper apply also if we consider federation tools that provide a representation of the data at the sources according to a different data model (e.g., XML). Obviously, depending on the specific data model adopted by the data federation tool, we have to resort to a suitable query language for expressing the source queries appearing in the mapping assertions. To adhere to our framework, the only constraint on this language is that it is able to extract tuples of values from the sources, a constraint that is trivially satisfied by virtually all query languages used in practice.

6.3 Extensions to the mapping language

As for the language used to express the mapping $\mathcal{M}$, we investigate the extension of the mapping language to allow for GLAV assertions, i.e., assertions that relate conjunctive

queries over the sources to conjunctive queries over the global schema. Such assertions are therefore an extension of both GAV and LAV mappings. Unfortunately, even with LAV mappings only, instance checking and query answering are no more in LOGSPACE wrt data complexity, as the following theorem shows.

Theorem 3. *Let $\mathcal{J} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a* MASTRO-I *data integration system extended with LAV mapping assertions, and D a database for $\mathcal{S}$. Answering a UCQ (in fact, a query constituted by a single atom) over $\mathcal{J}$ with respect to D is* NLOGSPACE*-hard in the size of D.*

Proof. The proof is again by a reduction from reachability in directed graphs. Let $G = \langle V, E \rangle$ be a directed graph, where V is the set of vertexes and E the set of directed edges. Again, we consider the graph represented through first-child and next-sibling functional relations F, N, S (cf. Figure 2).

We define the data integration system $\mathcal{J}_{lav} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ as follows:

- The alphabet of $\mathcal{G}$ consists of the atomic concept A_g and the atomic roles F_g, N_g, S_g, P_g, and $copy_g$. $\mathcal{G}$ consists only of the functionality assertions $\{(\mathsf{funct}\ \mathcal{R}_g) \mid \mathcal{R}_g \in \{F_g, N_g, S_g, P_g, copy_g\}\}$.
- $\mathcal{S}$ contains the binary relational tables F_s, N_s, and S_s, with columns c_1 and c_2, and the unary relational table A_s, with column c.
- The LAV mapping $\mathcal{M}$ is defined as follows (cf. Figure 4)[11]:

$$
\begin{array}{rcl}
A_s(x) & \rightsquigarrow & q_1(x) :- A_g(x), copy_g(x, x'), P_g(z, x), P_g(z, x') \\
\mathcal{R}_s(x, y) & \rightsquigarrow & q_2(x, y) :- \mathcal{R}_g(x, y), copy_g(x, x'), copy_g(y, y'), \mathcal{R}_g(x', y'), \\
 & & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{for } \mathcal{R} \in \{F, N, S\}
\end{array}
$$

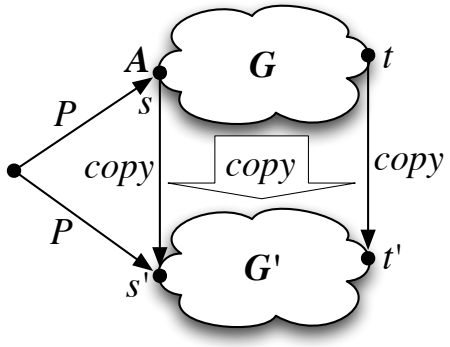


Fig. 4. Interpretation generated by the LAV mapping used in the proof of Theorem 3

[11] For simplicity, we do not include function symbols in the mapping since, as in the proof of Theorem 2, they would play no role in the reduction.

Then, from the graph G and the two vertexes s, t, we define the source database D_G as follows:

$$D_G \;=\; \{\mathcal{R}_s(a,b) \mid (a,b) \in \mathcal{R}, \text{ for } \mathcal{R} \in \{F,N,S\}\} \cup \{A_s(s)\}$$

Intuitively, D_G is simply constituted by the binary relations F_s, N_s, and S_s, used to represent the graph G, and a unary relation A_s containing s.

Now consider the query $Q(x) :- \; copy_g(x,x)\}$. Then, it is possible to show that t is reachable from s in G iff $t \in cert(Q, \mathcal{J}_{lav}, D_G)$. □

7 Conclusions

We close the paper by briefly mentioning some aspects that have been considered important for the problem of (ontology-based) data integration, but that have not been addressed in the present paper, and are left for future work on the system MASTRO-I.

A first important point is handling inconsistencies in the data, possibly using a declarative, rather than an adhoc procedural approach. An interesting proposal is the one of the INFOMIX system [27] for the integration of heterogeneous data sources (e.g., relational, XML, HTML) accessed through a relational global schema with powerful forms of integrity constraints. The query answering technique proposed in such a system is based on query rewriting in Datalog enriched with negation and disjunction, under stable model semantics [6, 21]. Some preliminary results on dealing with instance-level inconsistencies in a declarative way in *DL-Lite* ontologies can be found in [25].

A further aspect is that of instance level integration and mappings, which deals with the situation where individual instances, rather than ontology elements, in different sources need to be mapped to each other (cf., e.g, [24]).

Finally, one notable direction for further work is making MASTRO-I a "write-also" data integration tool. Indeed, while the present version of MASTRO-I provides support for answering queries posed to the data integration system, it is of interest to also deal with updates expressed on the global schema (e.g., according to the approach described in [13, 14]. The most challenging issue to be addressed in this context is to design mechanisms for correctly reformulating an update expressed over the ontology into a series of insert and delete operations on the data sources.

Acknowledgments. This research has been partially supported by the FET project TONES (Thinking ONtologiES), funded by the EU under contract number FP6-7603, by project HYPER, funded by IBM through a Shared University Research (SUR) Award grant, and by the MIUR FIRB 2005 project "Tecnologie Orientate alla Conoscenza per Aggregazioni di Imprese in Internet" (TOCAI.IT).

References

1. A. Acciarri, D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, M. Palmieri, and R. Rosati. QUONTO: QUerying ONTOlogies. In *Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005)*, pages 1670–1671, 2005.

2. B. Amann, C. Beeri, I. Fundulaki, and M. Scholl. Ontology-based integration of XML web resources. In *Proc. of the 1st Int. Semantic Web Conf. (ISWC 2002)*, pages 117–131, 2002.
3. C. Batini, M. Lenzerini, and S. B. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18(4):323–364, 1986.
4. P. A. Bernstein, F. Giunchiglia, A. Kementsietsidis, J. Mylopoulos, L. Serafini, and I. Zaihrayeu. Data management for peer-to-peer computing: A vision. In *Proc. of the 5th Int. Workshop on the Web and Databases (WebDB 2002)*, 2002.
5. A. Calì, D. Calvanese, G. De Giacomo, and M. Lenzerini. On the expressive power of data integration systems. In *Proc. of the 21st Int. Conf. on Conceptual Modeling (ER 2002)*, 2002.
6. A. Calì, D. Lembo, and R. Rosati. Query rewriting and answering under constraints in data integration systems. In *Proc. of the 18th Int. Joint Conf. on Artificial Intelligence (IJCAI 2003)*, pages 16–21, 2003.
7. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, and R. Rosati. Linking data to ontologies: The description logic *DL-Lite*$_A$. In *Proc. of the 2nd Workshop on OWL: Experiences and Directions (OWLED 2006)*, 2006.
8. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Data complexity of query answering in description logics. In *Proc. of the 10th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2006)*, pages 260–270, 2006.
9. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. *J. of Automated Reasoning*, 39(3):385–429, 2007.
10. D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, and R. Rosati. Data integration in data warehousing. *Int. J. of Cooperative Information Systems*, 10(3):237–271, 2001.
11. 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.
12. 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.
13. G. De Giacomo, M. Lenzerini, A. Poggi, and R. Rosati. On the update of description logic ontologies at the instance level. In *Proc. of the 21st Nat. Conf. on Artificial Intelligence (AAAI 2006)*, pages 1271–1276, 2006.
14. G. De Giacomo, M. Lenzerini, A. Poggi, and R. Rosati. On the approximation of instance level update and erasure in description logics. In *Proc. of the 22nd Nat. Conf. on Artificial Intelligence (AAAI 2007)*, pages 403–408, 2007.
15. 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.
16. 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.
17. J. Euzenat and P. Schwaiko. *Ontology Matching*. Springer, 2007.
18. 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.
19. M. R. Genereseth, A. M. Keller, and O. M. Duschka. Infomaster: An information integration system. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 539–542, 1997.

20. 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.
21. L. Grieco, D. Lembo, M. Ruzzi, and R. Rosati. Consistent query answering under key and exclusion dependencies: Algorithms and experiments. In *Proc. of the 14th Int. Conf. on Information and Knowledge Management (CIKM 2005)*, pages 792–799, 2005.
22. R. Hull. A survey of theoretical research on typed complex database objects. In J. Paredaens, editor, *Databases*, pages 193–256. Academic Press, 1988.
23. 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.
24. T. Kirsten and E. Rahm. BioFuice: mapping-based data integration in bioinformatics. In *Proc. of the 3rd Int. Workshop on Data Integration in the Life Sciences (DILS 2006),*, volume 4075 of *Lecture Notes in Computer Science*, pages 124–135. Springer, 2006.
25. D. Lembo and M. Ruzzi. Consistent query answering over description logic ontologies. In *Proc. of the 1st Int. Conf. on Web Reasoning and Rule Systems (RR 2007)*, 2007.
26. 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.
27. N. Leone, T. Eiter, W. Faber, M. Fink, G. Gottlob, G. Greco, E. Kalka, G. Ianni, D. Lembo, M. Lenzerini, V. Lio, B. Nowicki, R. Rosati, M. Ruzzi, W. Staniszkis, and G. Terracina. The INFOMIX system for advanced integration of incomplete and inconsistent data. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 915–917, 2005.
28. 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.
29. A. Y. Levy, D. Srivastava, and T. Kirk. Data model and query evaluation in global information systems. *J. of Intelligent Information Systems*, 5:121–143, 1995.
30. A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati. Linking data to ontologies. *J. on Data Semantics*, X:133–173, 2008.
31. A. Tomasic, L. Raschid, and P. Valduriez. Scaling access to heterogeneous data sources with DISCO. *IEEE Trans. on Knowledge and Data Engineering*, 10(5):808–823, 1998.
32. 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.