Discovering Functional Dependencies from Ontologies

Oscar Romero[1], Diego Calvanese[2], Alberto Abelló[1], and Mariano Rodríguez-Muro[2]

[1] Dept. de Llenguatges i Sistemes Informàtics, Univ. Pol. de Catalunya, Barcelona, Spain
{oromero|aabello}@lsi.upc.edu
[2] KRDB Research Centre, Free Univ. of Bozen-Bolzano, Bolzano, Italy
{calvanese|rodriguez}@inf.unibz.it

Abstract. Discovering functional dependencies is a fundamental step in the design of relational databases and in most system reengineering processes. Typically, this task has been performed over relational databases, at the logical or physical level. Those works addressing it at the logical level, often make some unrealistic assumptions (such as completeness of the data structures or similar names for semantically related attributes), while those addressing it at the physical level propose solutions that are computationally expensive, whose performance deteriorates with a large number of attributes or instances, and which cannot tolerate erroneous data. To overcome these limitations, while also better capturing data dependencies, we propose to rely instead on a conceptual representation of the domain of interest, specified in ER and formalized through a domain ontology expressed in the *DL-Lite* Description Logic. We propose an algorithm to discover functional dependencies from the domain ontology that exploits the inference capabilities of *DL-Lite*, thus fully taking into account the semantics of the domain. We also provide an evaluation of our approach in a real-world scenario.

1 Introduction

Discovering functional dependencies is a fundamental step in the design of relational databases and in most system reengineering processes, such as system maintainability and redesign. We can think of a well-formed relational schema as a set of attributes grouped according to functional dependencies. Nevertheless, designing large database schemas is a hard and time-consuming task far from being trivial, and functional dependencies provide information about the semantics of the domain that may be used to provide automated support in the design phase. Functional dependencies are also crucial for many reengineering processes, as the ones discussed below.

– System redesign: This is one of the traditional tasks of system maintainability. A bad choice of the schema may give rise to update, deletion, and insertion anomalies and redundant data storage that would not guarantee the integrity of data in the system. These problems have been thoroughly addressed in database theory by discovering functional dependencies and decomposing the schema appropriately according to the requirements (for instance, by normalizing the schema up to 3NF or BCNF) [1, 26].

– Schema / data integration: Merging different specifications that overlap in their domain of interest for giving a unified view of data stored in different systems is a typical reengineering process (for instance, after merging two companies). Discovering functional dependencies in each source to be integrated is a fundamental task, as functional dependencies facilitate matching the sources by looking for correspondences without considering design decisions such as denormalization [22].

– Improving query performance: At the physical level, functional dependencies give rise to different storage and access structures. For instance, if a functional dependency holds in a relation, it can be physically stored in decomposed form. Furthermore, the *chase* method is used for optimizing conjunctive queries in the presence of a large class of dependencies [1].

– Discovering aggregate measures of analysis: Either for data integration or analysis purposes, we may derive new valuable information by computing aggregate *measures* (in the sense of valuable data of interest for analysis purposes) in our domain of interest [21]. We may use functional dependencies to identify which *datatypes* (i.e., attributes) are functional dependencies of a given concept (and therefore, an aggregate measure of that concept).

– Multidimensional design of data warehouses: Data warehouses can be considered as a specific kind of data integration systems. Indeed, they homogenize and integrate data of organizations in a huge repository (the *data warehouse*), and we can think of the data warehouse schema as a view over the data source schemas. Furthermore, it is widely accepted that the conceptual schema of a data warehouse must be structured according to the *multidimensional model* [19]. As a consequence, the most common way to automatically look for subjects (i.e., facts) and dimensions of analysis is by discovering functional dependencies (since dimensions functionally depend of the fact) over the data sources [21].

Typically, all these tasks have been performed over relational databases, at the logical or physical level. Those works addressing it at the logical level, often make some unrealistic assumptions like completeness of the data structures or semantically related attributes that have similar names. A logical schema is tied to the design decisions made when devising the system (for instance, denormalization of data) and these decisions either made to fulfill the system requirements (for instance, improve query answering, avoid update/insertion/deletion anomalies, preserve features inherited from legacy systems, etc.) or naively made by non-expert users have a big impact on the data semantics captured in the schema. Thus, the assumption of completeness of data structures guarantees that no semantics are lost at the logical level, whereas the second assumption is addressed to identify potential foreign keys and referenced primary keys. Both assumptions are quite restrictive and easily not preserved in real world systems and that is the main reason why most of the approaches we may find in the literature work directly over the instances. Addressing this task at the physical level however, has other inherent problems. First, these approaches are only able to identify those dependencies holding in the data. Consequently, they cannot tolerate erroneous data, else the reliability of the functional dependencies identified would be affected. Finally, these approaches propose solutions that are computationally expensive and performance deteriorates with a large number of attributes or instances.

To overcome these limitations we propose to rely instead on a conceptual representation of the domain of interest, which is readily available for many systems built according to current software engineering practices. The software engineering area has claimed to use conceptual representations of the domain on top of systems to have an up-to-date and accurate formalization of the system domain. Therefore, although the best logical implementation according to the system requirements may lose semantic, we always have a clear understanding of the domain at the conceptual level. Specifically, we rely on conceptual schemas specified in ER or as UML class diagrams, and formalized through a domain ontology expressed in the *DL-Lite* Description Logic [6, 24]. We propose an algorithm to discover functional dependencies from the domain ontology that exploits the inference capabilities of *DL-Lite*, thus fully taking into account the semantics of the domain. We also provide an evaluation of our approach in a real-world scenario.

2 Conceptual modeling using description logics

Description Logics (DLs) [3] originate in the mid '80s to provide a formal basis to structured knowledge representation languages. We make use of *DL-Lite*$_\mathcal{A}$, a DL of the *DL-Lite* family [6, 5], which is particularly well suited for conceptual modeling due its ideal trade-off between expressive power and computational properties. We first present *DL-Lite*$_\mathcal{A}$, and then illustrate its modeling capabilities by means of an example.

2.1 A Tractable Description Logic: *DL-Lite*$_\mathcal{A}$

In DLs, objects with common properties are grouped into *concepts*, and the properties are represented through *roles*, denoting binary relations over the domain of interest. Complex concepts and roles are built inductively by starting from atomic ones (i.e., simply concept and role names) and applying a set of constructs. Different from traditional DLs, and following what done in other conceptual modeling formalisms such as UML class diagrams, *DL-Lite*$_\mathcal{A}$ distinguishes between (abstract) objects and (data) values. Hence, it distinguishes concepts, denoting sets of objects, from *value-domains*, denoting sets of values, and roles, denoting binary relations between objects, from *attributes*, denoting binary relations between objects and values. More precisely, concepts, roles, value-domains, and attributes in *DL-Lite*$_\mathcal{A}$ are formed starting from atomic elements according to the following syntax (where the distinction between *basic* and *arbitrary* elements is relevant in what follows):

	atomic	basic	arbitrary
concept	A	$B \longrightarrow A \mid \exists Q \mid \delta(U)$	$C \longrightarrow B \mid \neg B$
role	P	$Q \longrightarrow P \mid P^-$	$R \longrightarrow Q \mid \neg Q$
value-domain		$E \longrightarrow \rho(U)$	$F \longrightarrow \top_D \mid T_1 \mid \cdots \mid T_n$
attribute	U	$V \longrightarrow U$	$W \longrightarrow V \mid \neg V$

Above, $\delta(U)$ denotes the *domain* of U, i.e., the set of objects that U relates to values; $\rho(U)$ denotes the *range* of U, i.e., the set of values that U relates to objects; $\top_D$ is the universal value-domain; $T_1, \ldots, T_n$ are n pairwise disjoint unbounded value-domains, corresponding to data types, such as `string`, `integer`, etc. In the following, let $\mathsf{Inv}(Q) = P^-$ when $Q = P$, and $\mathsf{Inv}(Q) = P$ when $Q = P^-$.

In *DL-Lite*$_\mathcal{A}$, knowledge about the domain is represented by means of an *ontology* (or knowledge base), consisting of a TBox, encoding intensional knowledge, and an

ABox, encoding extensional knowledge on specific objects. Specifically, a *DL-Lite*$_{\mathcal{A}}$ TBox is constituted by a set of assertions of the form

$$B \sqsubseteq C, \qquad Q \sqsubseteq R, \qquad E \sqsubseteq F, \qquad V \sqsubseteq W, \qquad (\mathsf{funct}\ Q), \qquad (\mathsf{funct}\ U),$$

which respectively denote an inclusion between a basic and an arbitrary concept, role, value-domain, and attribute, and functionality on a role and on an attribute. As for the ABox, we introduce two disjoint alphabets, Γ_O of *object constants* denoting objects, and Γ_V of *value constants* denoting data values. A *DL-Lite*$_{\mathcal{A}}$ ABox is a finite set of *membership assertions* of the form (where $a, b \in \Gamma_O$ and $c \in \Gamma_V$):

$$A(a), \qquad P(a,b), \qquad U(a,c).$$

Definition 1. *A DL-Lite*$_{\mathcal{A}}$ *ontology* $\mathcal{O}$ *is a pair* $\langle \mathcal{T}, \mathcal{A} \rangle$*, where* $\mathcal{T}$ *is a DL-Lite*$_{\mathcal{A}}$ *TBox,* $\mathcal{A}$ *is a DL-Lite*$_{\mathcal{A}}$ *ABox, and the following conditions are satisfied:*

(1) for each atomic role P*, if either* $(\mathsf{funct}\ P)$ *or* $(\mathsf{funct}\ P^-)$ *occur in* $\mathcal{T}$*, then* $\mathcal{T}$ *does not contain assertions of the form* $Q \sqsubseteq P$ *or* $Q \sqsubseteq P^-$ *(for* Q *a basic role);*
(2) for each atomic attribute U*, if* $(\mathsf{funct}\ U)$ *occurs in* $\mathcal{T}$*, then* $\mathcal{T}$ *does not contain assertions of the form* $V \sqsubseteq U$ *(for* V *an atomic attribute);*

Intuitively, these two conditions say that, in a *DL-Lite*$_{\mathcal{A}}$ TBox, roles and attributes occurring in functionality assertions cannot be specialized. These conditions are crucial for the tractability of reasoning [24].

The semantics of *DL-Lite*$_{\mathcal{A}}$ is given in terms of FOL interpretations. An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a first order structure over the interpretation domain $\Delta^{\mathcal{I}}$ that is the disjoint union of $\Delta^{\mathcal{I}}_O$ and $\Delta^{\mathcal{I}}_V$, and of an *interpretation function* $\cdot^{\mathcal{I}}$ such that $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}_O$ for all $a \in \Gamma_O$, $c^{\mathcal{I}} \in \Delta^{\mathcal{I}}_V$ for all $c \in \Gamma_V$, and such that the following conditions are satisfied (below, $o, o' \in \Delta^{\mathcal{I}}_O$, and $v \in \Delta^{\mathcal{I}}_V$):

$$\begin{array}{rcl@{\qquad}rcl}
A^{\mathcal{I}} & \subseteq & \Delta^{\mathcal{I}}_O & P^{\mathcal{I}} & \subseteq & \Delta^{\mathcal{I}}_O \times \Delta^{\mathcal{I}}_O \\
(\exists Q)^{\mathcal{I}} & = & \{\, o \mid \exists o'.\, (o,o') \in Q^{\mathcal{I}} \,\} & (P^-)^{\mathcal{I}} & = & \{\, (o,o') \mid (o',o) \in P^{\mathcal{I}} \,\} \\
(\delta(U))^{\mathcal{I}} & = & \{\, o \mid \exists v.\, (o,v) \in U^{\mathcal{I}} \,\} & (\neg Q)^{\mathcal{I}} & = & (\Delta^{\mathcal{I}}_O \times \Delta^{\mathcal{I}}_O) \setminus Q^{\mathcal{I}} \\
(\neg B)^{\mathcal{I}} & = & \Delta^{\mathcal{I}}_O \setminus B^{\mathcal{I}} & & & \\
T_i^{\mathcal{I}} & \subseteq & \Delta^{\mathcal{I}}_V & U^{\mathcal{I}} & \subseteq & \Delta^{\mathcal{I}}_O \times \Delta^{\mathcal{I}}_V \\
\top_D^{\mathcal{I}} & = & \Delta^{\mathcal{I}}_V & (\neg V)^{\mathcal{I}} & = & (\Delta^{\mathcal{I}}_O \times \Delta^{\mathcal{I}}_V) \setminus V^{\mathcal{I}} \\
(\rho(U))^{\mathcal{I}} & = & \{\, v \mid \exists o.\, (o,v) \in U^{\mathcal{I}} \,\} & & &
\end{array}$$

We assume that the *unique name assumption* holds, i.e., different (object and value) constants are interpreted as different domain elements.

We define now when an interpretation $\mathcal{I}$ satisfies a TBox or ABox assertion. Specifically, $\mathcal{I}$ *satisfies*:

- $\alpha_1 \sqsubseteq \alpha_2$, if $\alpha_1^{\mathcal{I}} \subseteq \alpha_2^{\mathcal{I}}$;
- $(\mathsf{funct}\ \beta)$, where β is either P, P^-, or U, if $(o, e_1) \in \beta^{\mathcal{I}}$ and $(o, e_2) \in \beta^{\mathcal{I}}$ implies $e_1 = e_2$, for each $o \in \Delta^{\mathcal{I}}_O$, and e_1, e_2 in either $\Delta^{\mathcal{I}}_O$ or $\Delta^{\mathcal{I}}_V$;
- $A(a)$ if $a^{\mathcal{I}} \in A^{\mathcal{I}}$, $P(a, a')$ if $(a^{\mathcal{I}}, a'^{\mathcal{I}}) \in P^{\mathcal{I}}$, and $U(a, c)$ if $(a^{\mathcal{I}}, c^{\mathcal{I}}) \in U^{\mathcal{I}}$.

A *model* of an ontology $\mathcal{O}$ (resp., TBox $\mathcal{T}$) is an interpretation $\mathcal{I}$ that satisfies all assertions in $\mathcal{O}$ (resp., $\mathcal{T}$). An ontology $\mathcal{O}$ (resp., TBox $\mathcal{T}$) is *satisfiable* if it has at least one model, and $\mathcal{O}$ (resp., $\mathcal{T}$) *logically implies* an assertion α, denoted $\mathcal{O} \models \alpha$ (resp., $\mathcal{T} \models \alpha$), if α is satisfied in all models of $\mathcal{O}$ (resp., $\mathcal{T}$).

A conjunctive query (CQ) q over an ontology $\mathcal{O}$ is an expression of the form $q(\boldsymbol{x}) \leftarrow \mathit{body}(\boldsymbol{x}, \boldsymbol{y})$, where $\boldsymbol{x}$ are the so-called *distinguished variables*, $\boldsymbol{y}$ are the *non-distinguished* variables, and $\mathit{body}(\boldsymbol{x}, \boldsymbol{y})$ is a set of atoms of the form $A(x_o)$, $P(x_o, x'_o)$, $T_i(x_v)$, or $U(x_o, x_v)$, where x_o, x'_o are variables in $\boldsymbol{x}$ or $\boldsymbol{y}$ or constants in Γ_O, and x_v is a variable in $\boldsymbol{x}$ or $\boldsymbol{y}$ or a constant in Γ_V. Notice that CQs corresponds to SELECT-PROJECT-JOIN SQL queries, and hence are the queries most commonly posed to relational DBs. The query $q(\boldsymbol{x}) \leftarrow \mathit{body}(\boldsymbol{x}, \boldsymbol{y})$ is interpreted in $\mathcal{I}$ as the set $q^{\mathcal{I}}$ of tuples $\boldsymbol{e} \in \Delta^{\mathcal{I}} \times \cdots \times \Delta^{\mathcal{I}}$ such that, when we assign $\boldsymbol{e}$ to the variables $\boldsymbol{x}$, the first-order logic formula $\exists \boldsymbol{y}.\varphi(\boldsymbol{x}, \boldsymbol{y})$, where $\varphi(\boldsymbol{x}, \boldsymbol{y})$ is the conjunction of atoms in $\mathit{body}(\boldsymbol{x}, \boldsymbol{y})$, evaluates to true in $\mathcal{I}$. The reasoning service we are interested in is *(conjunctive) query answering*: given an ontology $\mathcal{O}$ and a query q over $\mathcal{O}$, return the *certain answers* to q over $\mathcal{O}$, i.e., all tuples $\boldsymbol{t}$ of elements of $\Gamma_V \cup \Gamma_O$ such that $\boldsymbol{t}^{\mathcal{I}} \in q^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{K}$, denoted $\mathcal{K} \models q(\boldsymbol{t})$.

As shown in [6, 24], all forms of inference over a *DL-Lite* ontology (e.g., satisfiability, logical implication, and CQ answering) can be done in polynomial time in the size of the TBox, and in AC_0[3] in the size of the ABox (i.e., w.r.t. data complexity [30]). In particular, to compute the certain answers of a CQ q, we can: (i) by using the assertions in the TBox, rewrite q to a union Q of CQs (which is directly translatable to an SQL query), and (ii) evaluate Q over the database corresponding to the ABox assertions. In this way, all forms of inference in *DL-Lite*$_{\mathcal{A}}$ can be carried out by exploiting standard commercial relational DB technology for manipulating the data (i.e., the ABox).

2.2 DLs and Conceptual Schemas

DLs share many similarities with representation formalisms used in different contexts, such as UML class diagrams[4] (UML-CDs), and the correspondence with these formalisms has been analyzed in detail [10, 4]. By virtue of this correspondence, automated inference algorithms developed for DLs can be used to reason over UML-CDs. Specifically, *DL-Lite*$_{\mathcal{A}}$ has been designed so as to capture the most important features of such diagrams, while keeping the complexity of inference low. We illustrate how a *DL-Lite*$_{\mathcal{A}}$ TBox can capture a UML-CD by means of an example, and refer to [8] for more details.

Consider the UML-CD in Figure 1. Intuitively, each UML class in the diagram is represented by an atomic concept in the TBox, each UML (binary)[5] association by an atomic role, and each UML attribute by an atomic attribute (we assume here that the name of the class is part of the attribute name). We describe how suitable TBox assertions capture the constraints imposed on the domain by the UML-CD.

[3] AC_0 is the complexity class that corresponds to the complexity in the size of the data of evaluating a first-order (i.e., SQL) query over a relational database (see, e.g., [1]).

[4] Our considerations apply, mutatis mutandis, also to ER schemas [11].

[5] Associations of arity greater than 2 can be handled through *reification* [10, 8].

- A generalization between two classes is represented by means of an inclusion assertion between the corresponding concepts, e.g., $\mathsf{Reservation} \sqsubseteq \mathsf{RentalAgreement}$.
- To represent domain and range of an association P, we use resp. $\exists P$ and $\exists P^-$. E.g., to represent that the $\mathsf{LocatedAt}$ association has Branch as domain and $\mathsf{Country}$ as range, we use $\exists \mathsf{LocatedAt} \sqsubseteq \mathsf{Branch}$ and $\exists \mathsf{LocatedAt}^- \sqsubseteq \mathsf{Country}$.
- To represent domain and range of an attribute U, we use resp. $\delta(U)$ and $\rho(U)$. E.g., to represent that $\mathsf{bestPrice}$ is an attribute of $\mathsf{RentalAgreement}$ with range Money, we use $\delta(\mathsf{bestPrice}) \sqsubseteq \mathsf{RentalAgreement}$ and $\rho(\mathsf{bestPrice}) \sqsubseteq \mathsf{Money}$.
- To capture mandatory participation in an association (i.e., a min. multiplicity 1), we use e.g., $\mathsf{Branch} \sqsubseteq \exists \mathsf{IsOfType}$ or $\mathsf{Customer} \sqsubseteq \exists \mathsf{Makes}^-$.
- To capture functionality of an association (i.e., a max. multiplicity 1), we use e.g., $(\mathsf{funct}\ \mathsf{LocatedAt})$ or $(\mathsf{funct}\ \mathsf{Makes}^-)$.
- Finally, to capture disjointness between classes, we use negation on concepts, as e.g., in $\mathsf{Car} \sqsubseteq \neg \mathsf{Branch}$.

By virtue of the reasoning capabilities of *DL-Lite*$_\mathcal{A}$ and of the encoding described above, inference over UML-CDs can be carried out by relying on the reasoning services provided by reasoners for *DL-Lite*$_\mathcal{A}$, e.g., by the system QuOnto [2, 25].

3 Discovering Functional Dependencies

In this section, we present our approach to discovering functional dependencies by relying on the assertions in an ontology.

We first recall some basic definitions regarding functional dependencies in the standard relational model (see, e.g., [1]). Consider a relation schema R, i.e., a relation symbol with an associated set of attributes, each denoting one component of R. A *functional dependency* (fd) over R has the form $R : X \rightarrow Y$, where X and Y are sets of attributes of R. We say that a relation r for R *satisfies* such a dependency if for each pair t_1, t_2 of tuples in r such that $\pi_X(t_1) = \pi_X(t_2)$, we have that $\pi_Y(t_1) = \pi_Y(t_2)$ (where, as usual, $\pi_X(t)$ denotes the projection of tuple t on the attributes in X).

It is well known (cf. [1]) that the following set of inference rules is sound and complete for implication of fds over a relation schema R (below, X, Y, and Z are sets of attributes of R, and juxtaposition of two sets stands for their union):
- If $Y \subseteq X$, then $R : X \rightarrow Y$ (*reflexivity*).
- If $R : X \rightarrow Y$, then $R : XZ \rightarrow YZ$ (*augmentation*).
- If $R : X \rightarrow Y$ and $R : Y \rightarrow Z$, then $R : X \rightarrow Z$ (*transitivity*).

In other words, all fds derived from a set $\mathcal{F}$ of fds over R, i.e., the $\mathcal{F}$*-closure*, can be computed by starting from $\mathcal{F}$ and exhaustively applying the above inference rules.

3.1 Functional dependencies over ontologies

We would like now to carry over to the conceptual level the standard notion of fd defined at the logical level. To this aim, we introduce the notion of fd over an ontology. We observe that previous work has already considered fds in the context of ontologies, see e.g., [9, 28, 29]. In these works, mimicking the notion in the relational model, a fd ensures that, if two objects that are instances of some concept share the same values for a set of attributes (or of attribute chains), then they share also the value of an additional attribute (or attribute chain), namely the attribute (chain) that functionally depends on the former attributes (chains). Instead, for the purposes described in Section 1, a fd should capture the intuition that the instances of one concept *functionally depend* on the instances of another concept. In other words, given two concepts[6] C_1 and C_2, we are interested in establishing whether each instance of C_1 allows one to determine a unique instance of C_2. We will denote this by $C_1 \rightarrow C_2$. Several observations are in order.

(i) The dependency between the two concepts C_1 and C_2 needs to be established explicitly, and this can be done by means of some role that relates C_1 to C_2. [7].

(ii) Since each instance of C_1 should determine a unique instance of C_2, and such a dependency is established through a role, we need to require such a role to be functional.

(iii) If we want to ensure a property analogous to transitivity (i.e., if $C_1 \rightarrow C_2$ and $C_2 \rightarrow C_3$, then also $C_1 \rightarrow C_3$), we need to allow the dependency to be established not only by atomic roles, but also by composite roles (i.e., role chains).

(iv) In an ontology, roles are not necessarily typed, i.e., they do not necessarily have a specified concept as domain and a specified concept as range. Therefore, one cannot establish in general that a role relates one concept to another concept. As a consequence, every untyped role would potentially allow one to establish that two arbitrary concepts are functionally dependent on each other, provided that the role relates one object to a single other object, i.e., that it is functional. This is clearly unsatisfactory, and therefore we need to enforce some stricter condition for a functional dependency $C_1 \rightarrow C_2$ to hold. Specifically, we will require not only that the role is functional, but also that the instances of C_1 mandatorily participate to the role, and that the role necessarily relates them to an instance of C_2.

The above observations lead us to the following definition of when a fd between two *DL-Lite*$_{\mathcal{A}}$ concepts holds in a given interpretation. We make use of the notion of *role chain* $Q_1 \circ \cdots \circ Q_n$ of basic roles, interpreted as the composition of the binary relations corresponding to the roles. Formally, for an interpretation $\mathcal{I}$, we have that

$$(Q_1 \circ \cdots \circ Q_n)^{\mathcal{I}} = Q_1^{\mathcal{I}} \circ \cdots \circ Q_n^{\mathcal{I}}.$$

We can then apply concept and role constructs to role chains (instead of basic roles), and the semantics naturally extends the one for the case of basic roles. When we talk about a role chain $Q_1 \circ \cdots \circ Q_n$ *over* a TBox $\mathcal{T}$, we intend that for each $i \in \{1, \ldots, n\}$, at least one of Q_i or $\mathsf{Inv}(Q_i)$ appears in $\mathcal{T}$. Similarly, a basic concept *over* $\mathcal{T}$ is any basic concept that can be constructed from atomic concepts and roles in $\mathcal{T}$.

[6] All our considerations can be easily extended to the case where C_2 is a value-domain instead of concept. We stick to pairs of concepts for space reasons.

[7] Note that in the relational model, attributes that functionally depend on other attributes are implicitly related through the relation schema to which the attributes belong.

Definition 2. *Given a DL-Lite$_\mathcal{A}$ TBox $\mathcal{T}$ and two concepts C_1 and C_2 over $\mathcal{T}$, the expression $C_1 \rightarrow C_2$ is called a* functional dependency *(over $\mathcal{T}$). Given an interpretation $\mathcal{I}$ of $\mathcal{T}$, we say that $C_1 \rightarrow C_2$* is satisfied in *$\mathcal{I}$, denoted $\mathcal{I} \models C_1 \rightarrow C_2$, if there is a role chain $S = Q_1 \circ \cdots \circ Q_n$ over $\mathcal{T}$, with $n > 0$ and such that for each object $o_1 \in C_1^\mathcal{I}$ there is exactly one object $o_2 \in C_2^\mathcal{I}$ such that $(o_1, o_2) \in S^\mathcal{I}$.*

Intuitively, the definition requires that each instance of C_1 determines a unique instance of C_2 by means of some chain of roles in $\mathcal{T}$. Note that the inverse of such a role chain corresponds to a path in the path-based identification constraints in [7].

From the above definition, it is immediate to verify that the following properties hold for fds over $\mathcal{T}$ involving concepts C_1, C_2, C_3, and for every interpretation $\mathcal{I}$:

Asserted:	If $\mathcal{I} \models (\mathsf{funct}\ Q)$, then $\mathcal{I} \models \exists Q \rightarrow \exists \mathsf{Inv}(Q)$.	(1)
Transitivity:	If $\mathcal{I} \models C_1 \rightarrow C_2$ and $\mathcal{I} \models C_2 \rightarrow C_3$, then $\mathcal{I} \models C_1 \rightarrow C_3$.	(2)
Left-inclusion:	If $\mathcal{I} \models C_1 \rightarrow C_2$ and $C_3^\mathcal{I} \subseteq C_1^\mathcal{I}$, then $\mathcal{I} \models C_3 \rightarrow C_2$.	(3)
Right-inclusion:	If $\mathcal{I} \models C_1 \rightarrow C_2$ and $C_2^\mathcal{I} \subseteq C_3^\mathcal{I}$, then $\mathcal{I} \models C_1 \rightarrow C_3$.	(4)

We are now interested in determining when an fd is logically implied by the assertions in the TBox, i.e., the fd is necessarily satisfied in all models of the TBox.

Definition 3. *Given a DL-Lite$_\mathcal{A}$ TBox $\mathcal{T}$, we say that a fd $C_1 \rightarrow C_2$ over $\mathcal{T}$ is logically implied by $\mathcal{T}$, denoted $\mathcal{T} \models C_1 \rightarrow C_2$, if $C_1 \rightarrow C_2$ is satisfied in every model of $\mathcal{T}$.*

In the following, we will restrict the attention to functional dependencies between basic concepts only, since in *DL-Lite$_\mathcal{A}$* negation is used only to assert disjointness. Exploiting the restrictions in the expressive power of *DL-Lite$_\mathcal{A}$*, we can show the following property.

Proposition 1. *Given a DL-Lite$_\mathcal{A}$ TBox $\mathcal{T}$ and two basic concepts B_1, B_2 over $\mathcal{T}$, we have that $\mathcal{T} \models B_1 \rightarrow B_2$ if and only if there is a role chain $S = Q_1 \circ \cdots \circ Q_n$ over $\mathcal{T}$, where $n > 0$, such that (i) $\mathcal{T} \models (\mathsf{funct}\ Q_i)$, for $i \in \{1, \ldots, n\}$, (ii) $\mathcal{T} \models B_1 \sqsubseteq \exists S$, and (iii) $\mathcal{T} \models \exists S^- \sqsubseteq B_2$.*

Proof (sketch). The "if" direction is a direct consequence of Definitions 2 and 3.

For the "only-if" direction, we observe that properties (ii) and (iii) follow from the canonical model property of the DLs of the *DL-Lite* family [6], and in particular of *DL-Lite$_\mathcal{A}$* [24]. Instead, for property (i), we exploit the tree-model property of *DL-Lite$_\mathcal{A}$*. This property is shared by most DLs [3], and states that if a TBox admits a model, then it admits one that has the structure of a tree (where the nodes of the tree are the elements of the interpretation domain, and the edges are determined by the role instances). In a tree-model, it is ruled out that from a given object o_1 there are two *different* paths labeled with the same roles that lead to the same object o_2. Hence, a chain of roles is forced to be functional in the TBox $\mathcal{T}$ only if also all of the component roles are forced to be functional. $\Box$

We can now exploit Proposition 1 to obtain a simple technique that derives pairs of concepts B_1, B_2 such that $\mathcal{T} \models B_1 \rightarrow B_2$. The technique is based on turning the properties (1)–(4) above into the following inference rules, which derive new fds from existing ones for a given TBox $\mathcal{T}$ and for basic concepts B_1, B_2, and B_3 over $\mathcal{T}$:

Asserted:	If $\mathcal{T} \models (\mathsf{funct}\ Q)$, then $\mathcal{T} \models \exists Q \rightarrow \exists \mathsf{Inv}(Q)$.	(5)
Transitivity:	If $\mathcal{T} \models B_1 \rightarrow B_2$ and $\mathcal{T} \models B_2 \rightarrow B_3$, then $\mathcal{T} \models B_1 \rightarrow B_3$.	(6)
Left-inclusion:	If $\mathcal{T} \models B_1 \rightarrow B_2$ and $\mathcal{T} \models B_3 \sqsubseteq B_1$, then $\mathcal{T} \models B_3 \rightarrow B_2$.	(7)
Right-inclusion:	If $\mathcal{T} \models B_1 \rightarrow B_2$ and $\mathcal{T} \models B_2 \sqsubseteq B_3$, then $\mathcal{T} \models B_1 \rightarrow B_3$.	(8)

We consider these rules to be applied exhaustively to all basic concepts over $\mathcal{T}$. Soundness of the rules follows directly from the corresponding properties above, while completeness is a consequence of Proposition 1. Moreover, since the number of basic concepts over $\mathcal{T}$ is finite, rule application clearly terminates.

We observe that the "left-inclusion" and "right-inclusion" rules propagate fds according to the TBox inclusion assertions, and as such they provide an interaction between functional and inclusion dependencies. In general, implication is undecidable when combining functional and inclusion dependencies [1], our setting is much simpler, since we only consider unary inclusion[8] and functional dependencies [12]. Note also that there is no counterpart of the augmentation rule for fd implication in the relational setting, since we deal only with unary functional dependencies.

4 An Algorithm to Discover Functional Dependencies in *DL-Lite*$_{\mathcal{A}}$

In this section, we propose an algorithm to discover all the fd's logically implied by a *DL-Lite*$_{\mathcal{A}}$ TBox $\mathcal{T}$, and which exploits the reasoning capabilities of a *DL-Lite*$_{\mathcal{A}}$ reasoner. Our algorithm starts from the asserted fds (see inference rule (5)), and then computes the closure of the asserted fds w.r.t. the remaining rules. We recall that the asserted fds are simply $\exists Q \rightarrow \exists \mathsf{Inv}(Q)$, for each functional role Q in the TBox.

The closure of the asserted fd's is computed as follows. First, we identify the sets $\mathcal{B}_d$ and $\mathcal{B}_r$ of all basic concepts that appear respectively in the domain and range of a functional basic role. To do so, we scan all functional basic roles, and for each such role Q and each basic concept B over $\mathcal{T}$, if $\mathcal{T} \models B \sqsubseteq \exists Q$ then we add B to $\mathcal{B}_d$, and if $\mathcal{T} \models B \sqsubseteq \exists \mathsf{Inv}(Q)$ then we add B to $\mathcal{B}_r$.

Then, for each pair of basic concepts $B_d \in \mathcal{B}_d$ and $B_r \in \mathcal{B}_r$, we need to check whether $\mathcal{T} \models B_d \sqsubseteq \exists S.B_r$, for some chain S of functional basic roles[9]. To perform such a check, we have to face the difficulty that in principle we have to try all possible lengths n of the chain S, and all the possible ways of composing it by means of functional basic roles. To tackle the latter issue, we introduce a new atomic role U in $\mathcal{T}$, and for each basic role Q such that $(\mathsf{funct}\ Q)$ is in $\mathcal{T}$, we add to $\mathcal{T}$ the assertion $Q \sqsubseteq U$[10]. Hence, U acts as a super-role of all functional basic roles in $\mathcal{T}$ and it is sufficient to consider S as a chain of U n times with itself, for suitable values of n. We then iterate over n until we have tried a sufficiently large value (see below).

[8] Note that, in *DL-Lite*, role inclusions are restricted so as not to inteeact with functionality.

[9] The concept $\exists S.B_r$ is called a qualified existential and is interpreted as $\{o \mid \exists o'.(o,o') \in S^{\mathcal{I}} \land o' \in B_r^{\mathcal{I}}\}$. It is not a *DL-Lite*$_{\mathcal{A}}$ concept.

[10] Note that this is compatible with the conditions in Definition 1.

In *DL-Lite*$_{\mathcal{A}}$ we cannot directly check the logical implication $\mathcal{T} \models B_d \sqsubseteq \exists S.B_r$, with $S = U \circ \cdots \circ U$ (for some fixed length n of the chain). However, we can easily encode such a check into the problem of computing the certain answers to the CQ

$$Q^n_{B_d,B_r}() \leftarrow B_d(a), U(a,x_1), U(x_1,x_2), \ldots, U(x_{n-1},x_n), B_r(x_n)$$

over the ABox constituted only by the assertion $B_d(a)$. Indeed, since the only fact in the ABox is one involving B_d, it is not possible to satisfy the atoms $U(x,x')$ and $B_r(x_n)$ with facts in the ABox. Hence, the only case in which the answer to the query could anyway be positive, is when the whole body of $Q^n_{B_d,B_r}$ can be rewritten to just $B_d(a)$ [6]. And this is precisely the case when $\mathcal{T} \models B_d \sqsubseteq \exists S.B_r$. Notice that we are taking advantage of the query rewriting technique for *DL-Lite*$_{\mathcal{A}}$, which exploits the knowledge contained in the TBox of the ontology to actually compute the right-inclusion and left-inclusion inference rules with the *DL-Lite*$_{\mathcal{A}}$ reasoner.

The question that we still need to address is which is the maximum bound for the length n of the role chain S. If the ontology does not contain functional cycles, we should stop when no new answer is retrieved. However, it is not uncommon to find functional cycles in a real world ontology. In this case, we should stop looking for functional dependencies originating at a concept B_d when (i) for a given length no results are provided, or (ii) no new concepts are proposed with regard to previous iterations. Intuitively, the reason is that in *DL-Lite* all the roles involved in a functional path must be functional as well, and hence at each step we must get, at least, one new concept of the longest path. Otherwise we are looping in a cycle.

More precisely, let B_d be the concept from where we start looking for functional dependencies and $\mathcal{D}_i$ the set of concepts that we have already identified up to iteration i. Let B_r be a concept functionally dependent on B_d and not yet identified.

– If B_d functionally identifies B_r, then there must be a role chain S' that connects $\mathcal{D}_i$ and B_r. Let $\mathcal{D}_?$ be the concepts along S'. Note that $\mathcal{D}_i$ and $\mathcal{D}_?$ are disjoint, since the $\mathcal{D}_i$ contains concepts already visited, while $\mathcal{D}_?$ not.
– At least, one concept of $\mathcal{D}_i$ and one concept of $\mathcal{D}_?$ must be directly related. Let's call them B_i and $B_?$, respectively.
– If along the $i+1$-th iteration we do not identify any new concept, then, $B_? \in \mathcal{D}_i$, which contradicts our initial assumption that $\mathcal{D}_i$ and $\mathcal{D}_?$ are disjoint.

4.1 Computational Complexity

An upper bound for the maximum number of queries we will have to pose is $\Theta(n \cdot |\mathcal{B}_d| \cdot |\mathcal{B}_r|)$, where n is length of the maximum chain of functional roles. However, this is an upper bound not reachable in practice because those concepts that do not get any new solution for paths of length i, are not queried in the next iterations. In most cases (considering ontologies in real applications), most of the concepts will end with n being rather small, as discussed in Section 6.1. Furthermore, if in a previous iteration we have shown that $\mathcal{T} \models B_d \rightarrow B_r$, then, this pair will not be checked again in next iterations.

We note that the computational complexity of the rewriting algorithm of *DL-Lite*$_{\mathcal{A}}$ is exponential in the size of the query. However, this turns out to be manageable for real ontologies, given that the number of times we have to concatenate role U is relatively small, cf. Section 6.1.

5 *DL-Lite$_{\mathcal{A}}$* Ontologies from Conceptual Schemas

The fds as introduced in Section 3 are conceived for arbitrary *DL-Lite$_{\mathcal{A}}$* ontologies. Nevertheless, there are some interesting additional considerations to be made regarding *DL-Lite$_{\mathcal{A}}$* ontologies derived from conceptual schemas. Consider the UML diagram depicted in Figure 1 and let $\mathcal{T}_{eu}$ be the corresponding *DL-Lite$_{\mathcal{A}}$* TBox. Specifically, the hasAssigned association results in the following assertions:

$$\begin{aligned} \exists \mathsf{hasAssigned} &\sqsubseteq \mathsf{RentalAgreement} \\ \exists \mathsf{hasAssigned}^- &\sqsubseteq \mathsf{Assignment} \\ \mathsf{Assigment} &\sqsubseteq \exists \mathsf{hasAssigned}^- \end{aligned} \qquad \begin{aligned} &(\mathsf{funct}\ \mathsf{hasAssigned}) \\ &(\mathsf{funct}\ \mathsf{hasAssigned}^-) \end{aligned}$$

According to Definition 3, we have that $\mathcal{T}_{eu} \models \mathsf{Assignment} \rightarrow \mathsf{RentalAgreement}$, but $\mathcal{T}_{eu} \not\models \mathsf{RentalAgreement} \rightarrow \mathsf{Assignment}$), since RentalAgreement does not have a mandatory participation in hasAssertion. As discussed in Section 3.1, the mandatory participation is needed in arbitrary *DL-Lite$_{\mathcal{A}}$* ontologies to avoid discovering meaningless functional dependencies. For instance, consider the following TBox $\mathcal{T}$:

$$\exists P_1 \sqsubseteq A_1 \quad \exists P_1^- \sqsubseteq A_2 \quad (\mathsf{funct}\ P_1) \qquad \exists P_2 \sqsubseteq A_3 \quad \exists P_2^- \sqsubseteq A_4 \quad (\mathsf{funct}\ P_2)$$

Without requiring the mandatory participation we would have that $\mathcal{T} \models A_1 \rightarrow A_4$. Indeed, both P_1 and P_2 are functional, and therefore, in every model of $\mathcal{T}$, every instance of A_1 is connected to at most one instance of A_4 via the role chain $P_1 \circ P_2$.

However this scenario cannot happen in ontologies derived from conceptual schemas. In an UML-CD (or ER schema) two classes are supposed to be disjoint unless they are related by a generalization relationship and furthermore, strict role-typing is assumed (i.e., exactly the opposite assumptions to those in arbitrary *DL-Lite$_{\mathcal{A}}$* ontologies). Hence, when translating UML-CDs to *DL-Lite$_{\mathcal{A}}$* it makes sense to identify functional dependencies from non mandatory relationships. With this aim, we redefine the functional property definition presented in Section 3.1 for *DL-Lite$_{\mathcal{A}}$* ontologies derived from conceptual schemas:

Definition 4. *Given a DL-Lite$_{\mathcal{A}}$ TBox $\mathcal{T}$, an atomic role P in $\mathcal{T}$ is* strict role-typed in *$\mathcal{T}$ if there is a single atomic concept A_1 such that $\exists P \sqsubseteq A_1$ is in $\mathcal{T}$, and a single atomic concept A_2 such $\exists P^- \sqsubseteq A_2$ is in $\mathcal{T}$. The concepts A_1 and A_2 are called respectively the* domain *and* range *of P. A DL-Lite$_{\mathcal{A}}$ TBox $\mathcal{T}_c$ is called DL-Lite$_{\mathcal{A}}$* conceptual schema *if each atomic role P is strict role-typed in $\mathcal{T}$ and for each pair of atomic concepts A_1, A_2, either A_1 and A_2 are disjoint (i.e., $\mathcal{T}_c \models A_1 \sqsubseteq \neg A_2$) or A_1 and A_2 participate in the same concept taxonomy (i.e., there is an atomic concept A such that $\mathcal{T}_c \models A_1 \sqsubseteq A$ and $\mathcal{T}_c \models A_2 \sqsubseteq A$).*[11]

Definition 5. *Given a DL-Lite$_{\mathcal{A}}$ conceptual schema $\mathcal{T}_c$, two basic concepts B_1 and B_2 over $\mathcal{T}_c$, and an interpretation $\mathcal{I}$ of $\mathcal{T}$, we say that $\mathcal{I} \models B_1 \rightarrow B_2$ if there is a chain $S = Q_1 \circ \cdots \circ Q_n$, with $n > 0$, of roles that are strict role-typed in $\mathcal{T}$, where B_1 is the domain of Q_1, B_2 is the range of Q_n, and such that for each object $o_1 \in \exists Q_1^{\mathcal{I}}$ there is exactly one object $o_2 \in C_2^{\mathcal{I}}$ such that $(o_1, o_2) \in S^{\mathcal{I}}$.*

[11] Note that the concept A may coincide with A_1 or A_2.

Roughly speaking, we may relax the mandatory participation of B_1 in S thanks to the implicit constraints we may find in a *DL-Lite* conceptual schema.

We can take advantage of the algorithm presented in Section 4 to discover functional dependencies over *DL-Lite*$_\mathcal{A}$ conceptual schemas by adding the following two assertions for each functional role P with domain A_1 and range A_2:

$$A_1 \sqsubseteq \exists P \qquad\qquad A_2 \sqsubseteq \exists P^-$$

Indeed, we are adding a mandatory participation for the role to its domain and range. In terms of UML-CDs, we are modifying the cardinality of the relationship and making it mandatory. With this trick we fulfill Definition 2 of fd and despite this change, the semantics with regard to fd's will not change and the results we get are sound. Notice that these assertions are only needed while discovering functional dependencies and they have to be retracted once the algorithm has finished.

This trick cannot be applied for arbitrary *DL-Lite* ontologies. In an arbitrary *DL-Lite*$_\mathcal{A}$ ontology disjointness of concepts cannot be assumed and therefore, adding the domain and range assertion we would modify the semantics of the model also with respect to fds. As a consequence, we could identify false fds.

6 Case Study

In this section we show results got after applying our algorithm over a real case study. Consider the conceptual diagram depicted in Figure 1. This diagram corresponds to a piece of the EU-Car Rental case study. It presents a car rental company with branches in several countries which provides typical rental services. This conceptual schema collects information about cars, branches, rental agreements, assignments, etc. A detailed specification of this case study may be found in [16].

In our approach, to compute the functional dependencies depicted in this diagram we will first need to transform the UML diagram into a *DL-Lite*$_\mathcal{A}$ ontology as explained in Section 2. Once it is done, we can launch our algorithm and compute the functional dependencies. It is remarkable the relevance of automating this process as in this small example we are able to discover 426 functional dependencies. The concept with more functional dependencies is closedrental with 44. For instance, an interesting concept to take a look is maintenancescheduled. This concept has 24 functional dependencies: datescheduled (its attribute), servicedepot (and its attributes: servicedepotname, servicedepotcapacity) and those inherited from owncar and needsmaintenance through the concept taxonomy it belongs to (i.e., branch and its attribute branchname; branchtype and its attribute branchtypename; country and its attributes countryname, countrymechanicalreq., countryemissionsreq. and cartax; carmodel and its attributes name and features; cargroup and its attribute cargroupname and those attributes inherited from owncar and car: id, currentmileage, mileagefromlastservice, lastmaintenancedate, acquisitiondate and available).

We would also like to remark that despite having many functional dependencies, it is easy to show them to the user. Many functional dependencies identified are datatypes. Moreover, given a concept *C*, its set of functional dependencies may be depicted as a tree with *C* as root node, and most of these trees will overlap. For instance, the fd tree of

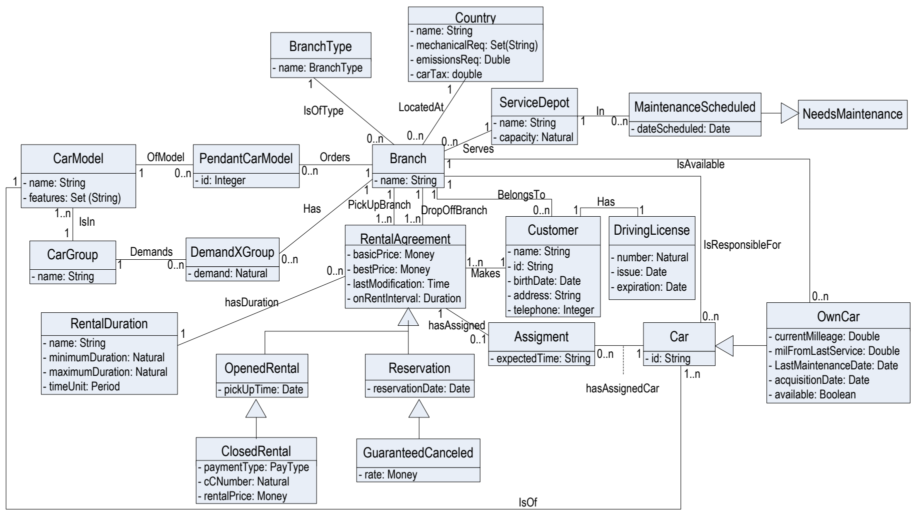

Fig. 1. The EU-Car Rental Case Study

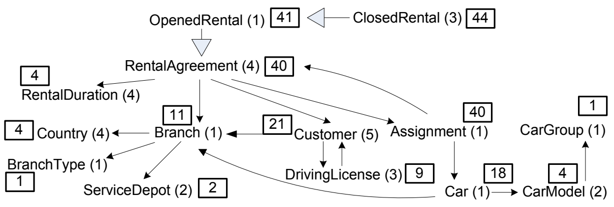

Fig. 2. A compacted view of the functional dependencies in the EU-Car Rental case study

branch will be a subtree within the rentalagreement fd tree. Figure 2 takes advantage of these properties to show the functional dependencies of closedrental (the concept with most fds in our example) and the functional dependencies of all those concepts in its tree; giving rise to a directed graph. This figure must be read as follows. Starting from a concept, following all the directed arrows we may get to all its functional dependencies (for instance, branch and its subtree are functional dependencies of customer, and the whole three of rentalagreement is a subtree of assignment. Closedrental and openedrental have two special arrows that stand for inheritance; thus, they get all the functional dependencies their parents have). Numbers in brackets represent the number of functional datatypes that concept has (since by definition, datatypes do not have dependencies we can overlook them at first sight and only show them if the user asks for). The squared number stands for the functional dependencies that concept has. Notice that we cannot compute the number of fds of a concept by adding the squared numbers of its sons, as some fds are shared between them. For instance, in the number of fds of customer we are considering the subtree of branch which is also directly related to rentalagreement, so when calculating the number of fds of rentalagreement this subtree must be counted just once. Finally, the reader must notice that if rentalduration has

4 functional dependencies (not shared with any other concept), rentalagreement will get 5 fds from this edge (4 fds plus rentalduration itself). In general, we need 4 graphs like this to depict the whole functional dependencies found in this example: the one shown in figure 2 and the analogous graphs for pendantcarmodel, demandpergroup and maintenancescheduled.

6.1 Statistics over the Case Study

For the sake of comprehension, we have discussed till now only a piece of the EU Car-Rental case study. We consider now the full EU Car-Rental case study of which we computed the closure of functional dependencies by applying the algorithm presented in Section 4. This case study has 65 concepts and 170 roles (30 of which are subsumption assertions between classes).

A total of 1908 functional dependencies were found. The total computation time was 2.332 seconds from which, 0.080 seconds were used by the reasoner to classify the ontology, 0.006 seconds were required to query for candidate domains and ranges for the functional paths (i.e., the $\mathcal{B}_d$ and $\mathcal{B}_r$ sets presented in Section 4, and the remaining time (2.246 seconds) was used to compute the functional dependencies.

To run these tests we used the FaCT++ reasoner [18]. We note that FaCT++ doesn't support answering conjunctive queries, As a workaround, we had to devise a subsumption verification query which was $true$ iff the CQ of Section 4 is non-empty, and $false$ otherwise. The query which complies with this specification is $(B_1 \sqsubseteq \exists U.\exists U.\cdots.B_2)$? were B_1 corresponds to the current domain to be tested, the number of nested U's corresponds to the number of U atoms in the original CQ and B_2 corresponds to the range of the functional path to be tested.

Furthermore, we also computed the functional paths (i.e., the composition of roles) that verified the query shown above. In order to do this, we sent additional queries to the reasoner, whenever we had verified the existence of a path of length n. In these queries we replaced the n'th occurance of role U in the qualified existential chain with each of the sub-roles of U. In this case, 41.039 seconds were spent pinning-down the specific roles which triggered the existence of these paths.

The computer used in these test was equipped with an Intel Core 2 Duo 2.16 GHz processor, 3 GB of RAM. With respect to software, we used the 64 bit version of the Java Runtime Environment 1.6 and the 64 bit version of the FaCT++ runtime binaries.

7 Related Work

Previous approaches for discovering functional dependencies over relational databases address this task either at the logical or physical level and therefore, cannot be smoothly compared to our approach. Addressing this task at a logical level entails that results got are tied to the design decisions made when devising the system. A logical schema may lack semantics regarding a conceptual schema. Logical design decisions such as overlooking foreign keys in the relational schema or denormalizing data (i.e., collapsing relations) directly impact on the quality of results got. For this reason, some approaches try to overcome the logical schema lack of semantics with additional semantics or assumptions. For instance, assuming that two semantically related attributes have similar names (i.e., with the same root, synonyms, etc.), we may look for potential foreign key - referenced primary keys by lexical matching of attribute names [17]. Another alternative is considering additional semantic sources such as database transactions [27] but in any case, these approaches introduce approximate patterns that must be asserted by sampling data. For this reason, most approaches in the literature address this task directly at a physical level [14, 15, 23, 20].

Works addressing this task at a physical level focus on generating the minimum number of functional dependencies hypotheses to be verified over data. Let N be the number of attributes in a relational schema, these approaches look for functional dependencies of the kind: $R: X \rightarrow A$, where A is a single attribute and X a set of attributes. Thus, the searching space has an exponential number of combinations at the LHS (i.e., 2^{N-1}) as hypotheses. All these works focus on introducing pruning rules that will reduce the combinations generated as hypotheses. For instance, by applying the *Armstrong axioms*. Given an initial set of functional dependencies $\mathcal{F}$, the Armstrong axioms only generate functional dependencies in the closure of $\mathcal{F}$ (i.e., $\mathcal{F}^+$). The main objective of previous works is to find a minimum set of functional dependencies $\mathcal{F}'$ such that $\mathcal{F}^+$ is equivalent to $\mathcal{F}'^+$. Said in other words $\mathcal{F}'$ is a *minimal cover* of F [1] (i.e., a reduced representative set of functional dependencies in $\mathcal{F}$). Nevertheless, these approaches are only able to identify those dependencies holding in data (since in the end, they are generating hypotheses to be validated over data) and consequently, they cannot easily tolerate erroneous data [13]. Otherwise, erroneous data may generate functional dependencies that do not hold or overlook real ones. Finally, all these approaches propose solutions that are computationally expensive and their performance deteriorates with a large number of attributes or instances. Moreover, once we have generated our reduced set of hypotheses we still need to verify them querying data that for large number of hypotheses is expensive.

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison Wesley, 1995.
2. A. Acciarri, D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, M. Palmieri, and R. Rosati. QUONTO: QUerying ONTOlogies. In *Proc. of AAAI 2005*, 2005.
3. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.
4. D. Berardi, D. Calvanese, and G. De Giacomo. Reasoning on UML class diagrams. *Artificial Intelligence*, 168(1–2):70–118, 2005.
5. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Data complexity of query answering in description logics. In *Proc. of KR 2006*, pages 260–270, 2006.
6. 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.
7. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Path-based identification constraints in description logics. In *Proc. of KR 2008*, pages 231–241, 2008.

8. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Conceptual modeling for data integration. In A. Borgida, V. Chaudhri, P. Giorgini, and E. Yu, editors, *Essays in Honour of John Mylopoulos*, LNCS. Springer, 2009. To appear. Available at `http://www.inf.unibz.it/~calvanese/papers-html/book-mylopoulos-2009.html`.
9. D. Calvanese, G. De Giacomo, and M. Lenzerini. Identification constraints and functional dependencies in description logics. In *Proc. of IJCAI 2001*, pages 155–160, 2001.
10. D. Calvanese, M. Lenzerini, and D. Nardi. Description logics for conceptual data modeling. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems*, pages 229–264. Kluwer Academic Publishers, 1998.
11. P. P. Chen. The Entity-Relationship model: Toward a unified view of data. *ACM Trans. on Database Systems*, 1(1):9–36, Mar. 1976.
12. S. S. Cosmadakis, P. C. Kanellakis, and M. Vardi. Polynomial-time implication problems for unary inclusion dependencies. *J. of the ACM*, 37(1):15–46, Jan. 1990.
13. J. Demetrovics, G. O. Katona, and D. Miklós. Functional Dependencies Distorted by Errors. *Discrete Applied Mathematics*, 156(6):862–869, 2008.
14. J. Demetrovics and V. D. Thi. Some remarks on generating armstrong and inferring functional dependencies relation. *Acta Cybern.*, 12(2):167–180, 1995.
15. P. A. Flach and I. Savnik. Database dependency discovery: A machine learning approach. *AI Communications*, 12(3):139–160, 1999.
16. L. Frías, A. Queralt, and A. Olivé. EU-Rent Car Rentals Specification. Technical report, Departament de Llenguatges i Sistemes Informatics, 2003.
17. J.-L. Hainaut, M. Chandelon, C. Tonneau, and M. Joris. Contribution to a theory of database reverse engineering. In *Proc. of the 1st Working Conf. on Reverse Engineering*, pages 161–170. IEEE, 1993.
18. I. Horrocks. Using an expressive description logic: FaCT or fiction? In *Proc. of KR'98*, pages 636–647, 1998.
19. W. H. Inmon. *Building the Data Warehouse*. John Wiley & Sons, 1990.
20. M. Kantola, H. Mannila, K.-J. Räihä, and H. Siirtola. Discovering Functional and Inclusion Dependencies in Relational Databases. *Int. J. of Intelligent Systems*, 7:591–607, 1992.
21. R. Kimball, L. Reeves, W. Thornthwaite, and M. Ross. *The Data Warehouse Lifecycle Toolkit: Expert Methods for Designing, Developing and Deploying Data Warehouses*. John Wiley, 1998.
22. M. Lenzerini. Data integration: A theoretical perspective. In *Proc. of PODS 2002*, pages 233–246, 2002.
23. W. M. Lim. Discovery of Constraints from Data for Information System Reverse Engineering. In *Proc. of the 2nd Australian Software Engineering Conf.*, pages 39–48. IEEE, 1997.
24. 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.
25. A. Poggi, M. Rodriguez, and M. Ruzzi. Ontology-based database access with DIG-Mastro and the OBDA Plugin for Protégé. In K. Clark and P. F. Patel-Schneider, editors, *Proc. of OWLED 2008 DC*, 2008.
26. R. Ramakrishnan and J. Gehrke. *Database Management Systems*. McGraw Hill, 2003.
27. H. B. K. Tan and Y. Zhao. Automated elicitation of functional dependencies from source codes of database transactions. *Information & Software Technology*, 46(2):109–117, 2004.
28. D. Toman and G. E. Weddell. On the interaction between inverse features and path-functional dependencies in description logics. In *Proc. of IJCAI 2005*, pages 603–608, 2005.
29. D. Toman and G. E. Weddell. On keys and functional dependencies as first-class citizens in description logics. *J. of Automated Reasoning*, 40(2–3):117–132, 2008.
30. M. Y. Vardi. The complexity of relational query languages. In *Proc. of STOC'82*, pages 137–146, 1982.