Conjunctive Query Containment in Description Logics with n-ary Relations

Diego Calvanese and **Giuseppe De Giacomo** and **Maurizio Lenzerini**
Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, 00198 Roma, Italy
{calvanese,degiacomo,lenzerini}@dis.uniroma1.it

Abstract

Recent research points out that query containment is a central problem in several database and knowledge base applications, including data warehousing, data integration, query optimization, and (materialized) view maintenance. In this paper we present a decision procedure for containment of conjunctive queries defined over a database schema specified in a very expressive description logic, comprising n-ary relations and general inclusion axioms on both concepts and relations.

1 Introduction

Query containment is the problem of checking whether for every data (or knowledge) base, the result of one query is always a subset of the result of another query. Many recent papers point out that query containment is a central problem in several database and knowledge base applications, including data warehousing, data integration, query optimization, (materialized) view maintenance, etc. (see for example [11, 1, 10]).

The issue of developing algorithms for query containment has been addressed by both the Database and Knowledge Representation community. Most of the results in databases deal with a framework where the database schema is expressed in a very simple data model (e.g. the relational model), and the query has the form of a conjunction of atomic queries (each involving one relation). On the other hand, the research on Knowledge Representation, and on Description Logics (DLs) in particular, typically assumes a more powerful schema specification language (although restricted to unary and binary predicates), and a query language where n-ary relations are either not used (e.g. queries as concepts) [2], or treated separately from the concepts and the roles of the schema (and therefore are not part of the knowledge base) [8, 9].

In this paper we present a new decision procedure for query containment which refers to a general framework where:

- The database (or knowledge base) schema is specified in a very expressive DL, comprising n-ary relations and general inclusion axioms on both concepts and relations;
- The query is a conjunctive query over the predicates used in the schema specification;
- Query containment amounts to checking whether in every model (database instance) of the schema, the set of tuples satisfying one query is a subset of the set of tuples satisfying another query.

The DL used in this work is able to capture virtually all data models (at least the structural part of them) we know of. In particular, our DL can express typed inclusion dependencies, existence dependencies, some forms of functional dependencies (at least those that are naturally expressed in an object-oriented framework) and many other forms of constraints.

We believe that our framework has several advantages, in that it provides significant expressive power for both specifying the schema, and for formulating queries. Our result shows that, in this powerful setting, query containment is still decidable in exponential time with respect to the size of the schema.

We are presently working on applying our results to a new approach to data integration, which we are developing in an Esprit project on Data Warehousing (called Data Warehouse Quality - DWQ).

2 The Logic $\mathcal{DLR}$

We introduce the description logic $\mathcal{DLR}$, which includes *concepts* and *n-ary relations*. $\mathcal{DLR}$ is inspired by the languages introduced in [4, 5, 6, 3], where the notion of n-ary relation was already present. We believe that $\mathcal{DLR}$ is a natural extension of DLs towards n-ary relations.

We assume to deal with a finite set of atomic relations and concepts, denoted by $\mathbf{P}$ and A respectively. We use $\mathbf{R}$ to denote arbitrary relations (of given arity between 2 and n_{max}) and C to denote arbitrary concepts, respectively built according to the following syntax:

$$\begin{array}{rcl} \mathbf{R} & ::= & \mathbf{P} \mid \$i/n\!:\!C \mid \neg\mathbf{R} \mid \mathbf{R}_1 \sqcap \mathbf{R}_2 \\ C & ::= & A \mid \neg C \mid C_1 \sqcap C_2 \mid \\ & & \forall\mathbf{R}_1[\$i].\mathbf{R}_2 \mid \exists^{\geq k}\mathbf{R}_1[\$i].\mathbf{R}_2 \end{array}$$

A relation of the form $\$i/n\!:\!C$ has arity n.

$$\begin{array}{rcl}
\mathit{fol}(\mathbf{P}) & = & \lambda\vec{\mathbf{x}}.\mathbf{P}(\vec{\mathbf{x}}) \\
\mathit{fol}(\$i/n\!:\!C) & = & \lambda\vec{\mathbf{x}}.\mathit{rel}_n(\vec{\mathbf{x}}) \wedge \mathit{fol}(C)(x_i) \\
\mathit{fol}(\neg\mathbf{R}) & = & \lambda\vec{\mathbf{x}}.\mathit{rel}_n(\vec{\mathbf{x}}) \wedge \neg\mathit{fol}(\mathbf{R})(\vec{\mathbf{x}}) \\
\mathit{fol}(\mathbf{R}_1 \sqcap \mathbf{R}_2) & = & \lambda\vec{\mathbf{x}}.\mathit{fol}(\mathbf{R}_1)(\vec{\mathbf{x}}) \wedge \mathit{fol}(\mathbf{R}_2)(\vec{\mathbf{x}}) \\
\\
\mathit{fol}(A) & = & \lambda x.A(x) \\
\mathit{fol}(\neg C) & = & \lambda x.\neg\mathit{fol}(C)(x) \\
\mathit{fol}(C_1 \sqcap C_2) & = & \lambda x.\mathit{fol}(C_1)(x) \wedge \mathit{fol}(C_2)(x) \\
\mathit{fol}(\forall\mathbf{R}_1[\$i].\mathbf{R}_2) & = & \lambda x.\forall\vec{\mathbf{x}}.\mathit{fol}(\mathbf{R}_1)(\vec{\mathbf{x}}) \wedge x = x_i \supset \mathit{fol}(\mathbf{R}_2)(\vec{\mathbf{x}}) \\
\mathit{fol}(\exists^{\geq k}\mathbf{R}_1[\$i].\mathbf{R}_2) & = & \lambda x.\exists^{\geq k}\vec{\mathbf{x}}.x = x_i \wedge \mathit{fol}(\mathbf{R}_1)(\vec{\mathbf{x}}) \wedge \mathit{fol}(\mathbf{R}_2)(\vec{\mathbf{x}})
\end{array}$$

Figure 1: The semantics of $\mathcal{DLR}$

We consider only concepts and relations that are *well-typed*. An atomic relation is always well-typed. A relation of the form $\$i/n\!:\!C$ is well-typed if $i \in \{1,\ldots,n\}$ and C is well-typed. A relation $\mathbf{R}$ is well-typed if all atomic relations and all relations of the form $\$i/n\!:\!C$ occurring in $\mathbf{R}$ are well typed and have the same arity n (which is also the arity of $\mathbf{R}$). An atomic concept is always well-typed. A concept of the form $\forall\mathbf{R}_1[\$i].\mathbf{R}_2$ (similarly for $\exists^{\geq k}\mathbf{R}_1[\$i].\mathbf{R}_2$) is well-typed if $\mathbf{R}_1$ and $\mathbf{R}_2$ are well-typed, $\mathit{arity}(\mathbf{R}_1) = \mathit{arity}(\mathbf{R}_2) = n$, and $i \in \{1,\ldots,n\}$. Finally a concept is well-typed, if all concepts occurring in it are well-typed.

We give the semantics of our language by providing a translation $\mathit{fol}(\cdot)$ of concepts and relations to open formulae of first order logic with equality, and relying on usual first order interpretations.

We use lambda notation for open formulae, hence interpreting $E = \lambda\vec{\mathbf{x}}.\gamma(\vec{\mathbf{x}})$ as a predicate of arity n, where n is the number of variables of $\vec{\mathbf{x}}$. By the application $E(\vec{\mathbf{y}})$ we denote $\gamma(\vec{\mathbf{y}})$. We also introduce the abbreviation $\mathit{rel}_n(\vec{\mathbf{x}}) \doteq \mathbf{P}_1^n(\vec{\mathbf{x}}) \vee\cdots\vee \mathbf{P}_l^n(\vec{\mathbf{x}})$, where $\{\mathbf{P}_1^n,\ldots,\mathbf{P}_l^n\}$ is the set of all atomic relations of arity n. Let $\mathbf{P},\mathbf{R},\mathbf{R}_1,\mathbf{R}_2$ be of arity n. The semantics of the constructors of $\mathcal{DLR}$ is shown in Figure 1.

Given a first order interpretation $\mathcal{I}$ with domain $\Delta^{\mathcal{I}}$, a formula defined by the lambda expression $\lambda\vec{\mathbf{x}}.\gamma(\vec{\mathbf{x}})$ is interpreted as

$$(\lambda\vec{\mathbf{x}}.\gamma(\vec{\mathbf{x}}))^{\mathcal{I}} = \{\vec{\mathbf{d}} \in \Delta^{\mathcal{I}} \mid \gamma^{\mathcal{I},[\vec{\mathbf{x}}/\vec{\mathbf{d}}]} \text{ is true}\}$$

Hence the set $C^{\mathcal{I}}$ of instances of C in $\mathcal{I}$ is $\mathit{fol}(C)^{\mathcal{I}}$. Similarly, the set $R^{\mathcal{I}}$ of tuples that are instances of R is $\mathit{fol}(R)^{\mathcal{I}}$.

Intuitively, $\$i$ is used to denote the i-th component of a tuple [1]. We observe that "ordinary" DL roles can be modeled by binary relations, and inverse roles are expressed simply by specifying components in the appropriate order. Negated relations essentially are able to capture difference of relations, since in the semantics the extension of $\neg\mathbf{R}$ is obtained as the difference between the union of the extensions of all atomic relations of the same certain arity as $\mathbf{R}$ and the extension of $\mathbf{R}$.

[1] $\$i/n$ makes the arity of the tuple explicit.

A $\mathcal{DLR}$ *knowledge base* is constituted by a finite set of *inclusion assertions* of the form

$$\begin{array}{rcl}
\mathbf{R}_1 & \sqsubseteq & \mathbf{R}_2 \\
C_1 & \sqsubseteq & C_2
\end{array}$$

where $\mathbf{R}_1$ and $\mathbf{R}_2$ are of the same arity. We give also the semantics of assertions in terms of $\mathit{fol}(\cdot)$, by deriving the corresponding first order closed formulae:

$$\begin{array}{rcl}
\mathit{fol}(\mathbf{R}_1 \sqsubseteq \mathbf{R}_2) & = & \forall\vec{\mathbf{x}}.\mathit{fol}(\mathbf{R}_1)(\vec{\mathbf{x}}) \supset \mathit{fol}(\mathbf{R}_2)(\vec{\mathbf{x}}) \\
\mathit{fol}(C_1 \sqsubseteq C_2) & = & \forall x.\mathit{fol}(C_1)(x) \supset \mathit{fol}(C_2)(x)
\end{array}$$

The first order translation $\mathit{fol}(\mathcal{K})$ of a knowledge base $\mathcal{K}$ is the conjunction of the formulae obtained by translating the assertions of $\mathcal{K}$. A *model* $\mathcal{I}$ of $\mathcal{K}$ is a first order interpretation that satisfies $\mathit{fol}(\mathcal{K})$.

A *conjunctive query* q for a $\mathcal{DLR}$ knowledge base $\mathcal{K}$ is an open formula of the form

$$\lambda\vec{\mathbf{x}}.\exists\vec{\mathbf{y}}.\alpha(\vec{\mathbf{x}},\vec{\mathbf{y}})$$

where $\alpha(\vec{\mathbf{x}},\vec{\mathbf{y}})$ is a conjunction of atomic formulae whose predicates are atomic relations and concepts of $\mathcal{K}$, having $\vec{\mathbf{x}},\vec{\mathbf{y}}$ as arguments. The *arity* of the conjunctive query is equal to the number of variables of $\vec{\mathbf{x}}$.

We observe that, since assertions on both relations and concepts can be expressed in $\mathcal{K}$, every predicate used in the conjunctive query can actually denote any complex relation or concept expressible in $\mathcal{DLR}$. This distinguishes our approach with respect to [8, 9], where relations appearing in queries are not part of the knowledge base.

Given two conjunctive queries (of the same arity n) q_1 and q_2 for $\mathcal{K}$, we say that q_1 is *contained in* q_2 wrt to $\mathcal{K}$, if

$$\mathit{fol}(\mathcal{K}) \models \forall\vec{\mathbf{x}}.q_1(\vec{\mathbf{x}}) \supset q_2(\vec{\mathbf{x}}).$$

3 Deciding Conjunctive Query Containment

The decision problem we address is that of deciding whether a conjunctive query q_1 is contained in a conjunctive query q_2 wrt a $\mathcal{DLR}$ knowledge base $\mathcal{K}$. Let $q_1 = \lambda\vec{\mathbf{x}}.\exists\vec{\mathbf{y}}.\alpha(\vec{\mathbf{x}},\vec{\mathbf{y}})$ and $q_2 = \lambda\vec{\mathbf{x}}.\exists\vec{\mathbf{z}}.\beta(\vec{\mathbf{x}},\vec{\mathbf{z}})$ be two conjunctive queries for $\mathcal{K}$. Then q_1 is contained in q_2, i.e. $fol(\mathcal{K}) \models \forall\vec{\mathbf{x}}.(\exists\vec{\mathbf{y}}.\alpha(\vec{\mathbf{x}},\vec{\mathbf{y}}) \supset \exists\vec{\mathbf{z}}.\beta(\vec{\mathbf{x}},\vec{\mathbf{z}}))$, iff $fol(\mathcal{K}) \wedge \exists\vec{\mathbf{x}}.\exists\vec{\mathbf{y}}.\alpha(\vec{\mathbf{x}},\vec{\mathbf{y}}) \wedge \neg\exists\vec{\mathbf{z}}.\beta(\vec{\mathbf{x}},\vec{\mathbf{z}})$ is unsatisfiable, iff

$$fol(\mathcal{K}) \wedge \alpha(\vec{\mathbf{a}},\vec{\mathbf{b}}) \wedge \neg\exists\vec{\mathbf{z}}.\beta(\vec{\mathbf{a}},\vec{\mathbf{z}}) \qquad (1)$$

is unsatisfiable, where $\vec{\mathbf{a}}$, $\vec{\mathbf{b}}$ are distinct constants.

We develop an algorithm for deciding conjunctive query containment which is based on a reduction to unsatisfiability[2] of a concept C' in a knowledge base $\mathcal{K}'$ expressed in the EXPTIME-decidable DL $\mathcal{CIQ}$ [7].

We encode the three parts of formula (1) in $\mathcal{K}' = \mathcal{K}'_{\mathcal{K}} \cup \mathcal{K}'_{\alpha} \cup \mathcal{K}'_{\beta} \cup \mathcal{K}'_{aux}$.

Encoding of $\mathcal{K}$

$\mathcal{K}'_{\mathcal{K}}$ is the translation of the $\mathcal{DLR}$ knowledge base $\mathcal{K}$ into a $\mathcal{CIQ}$ knowledge base. Let us define a mapping $\sigma(\cdot)$ as follows:

$$\begin{array}{rcl}
\sigma(\mathbf{P}) & = & A_{\mathbf{P}} \\
\sigma(\$i/n\!:\!C) & = & \top_n \sqcap \exists f_i.\sigma(C) \\
\sigma(\neg\mathbf{R}) & = & \top_n \sqcap \neg\sigma(\mathbf{R}) \\
\sigma(\mathbf{R}_1 \sqcap \mathbf{R}_2) & = & \sigma(\mathbf{R}_1) \sqcap \sigma(\mathbf{R}_2) \\
\sigma(A) & = & A \\
\sigma(\neg C) & = & \top_1 \sqcap \neg\sigma(C) \\
\sigma(C_1 \sqcap C_2) & = & \sigma(C_1) \sqcap \sigma(C_2) \\
\sigma(\forall\mathbf{R}_1[\$i].\mathbf{R}_2) & = & \forall f_i^-.\neg\sigma(\mathbf{R}_1) \sqcup \sigma(\mathbf{R}_2) \\
\sigma(\exists^{\geq k}\mathbf{R}_1[\$i].\mathbf{R}_2) & = & \exists^{\geq k} f_i^-.\sigma(\mathbf{R}_1) \sqcap \sigma(\mathbf{R}_2) \\
\sigma(C_1 \sqsubseteq C_2) & = & \sigma(C_1) \sqsubseteq \sigma(C_2) \\
\sigma(\mathbf{R}_1 \sqsubseteq \mathbf{R}_2) & = & \sigma(\mathbf{R}_1) \sqsubseteq \sigma(\mathbf{R}_2)
\end{array}$$

where $A_{\mathbf{P}}$, A, $\top_1,\ldots,\top_{n_{max}}$ are newly introduced atomic concepts and $f_1,\ldots,f_{n_{max}}$ are newly introduced atomic roles. $\mathcal{K}'_{\mathcal{K}}$ contains the mapping $\sigma(\cdot)$ of all assertions in $\mathcal{K}$ and the following assertions ($\top$ is interpreted as the whole domain):

$$\begin{array}{rcll}
\top & \sqsubseteq & \top_1 \sqcup \cdots \sqcup \top_{n_{max}} & \\
\top & \sqsubseteq & \neg\exists^{\geq 2} f_i.\top & \text{for all } i \text{ s.t. } 1 \leq i \leq n_{max} \\
\forall f_i.\bot & \sqsubseteq & \forall f_{i+1}.\bot & \text{for all } i \text{ s.t. } 1 \leq i < n_{max} \\
\top_n & \equiv & \exists f_1.\top_1 \sqcap \cdots \sqcap \exists f_n.\top_1 \sqcap \forall f_{n+1}.\bot & \\
 & & \text{for each } n = arity(\mathbf{P}) \text{ for some } \mathbf{P} & \\
A_{\mathbf{P}} & \sqsubseteq & \top_n & \text{for each } \mathbf{P} \text{ of arity } n \\
A & \sqsubseteq & \top_1 & \text{for each concept name } A \text{ in } \mathcal{K}
\end{array}$$

The size of $\mathcal{K}'_{\mathcal{K}}$ is polynomial in the size of $\mathcal{K}$.

Intuitively, $\mathcal{K}'_{\mathcal{K}}$ is based on the *reification* of n-ary relations, i.e tuples are represented by individuals having one functional link f_i for each component. The correctness of this technique is due to the inability in $\mathcal{CIQ}$ of expressing that two chains of links meet the same individual, thus disallowing the possibility of creating two individuals representing the same tuple [5].

[2]A concept is *satisfiable* in a knowledge base $\mathcal{K}$ if $\mathcal{K}$ admits a model in which the concept has a nonempty extension.

Encoding of $\alpha(\vec{\mathbf{a}},\vec{\mathbf{b}})$

$\mathcal{K}'_{\alpha}$ encodes $\alpha(\vec{\mathbf{a}},\vec{\mathbf{b}})$, which is of the form

$$\mathbf{P}_1(\vec{\mathbf{c}}_{\mathbf{P}_1}) \wedge \cdots \wedge \mathbf{P}_p(\vec{\mathbf{c}}_{\mathbf{P}_p}) \wedge A_1(c_{A_1}) \wedge \cdots \wedge A_a(c_{A_a}).$$

In $\mathcal{K}'_{\alpha}$ we make use of special concepts, called *sk-concepts*, for representing the constants in $\alpha(\vec{\mathbf{a}},\vec{\mathbf{b}})$ (the properties of sk-concepts are specified by $\mathcal{K}'_{aux}$ – see later). More specifically, we introduce one sk-concept c for each constant c in $\alpha(\vec{\mathbf{a}},\vec{\mathbf{b}})$, and one sk-concept $\vec{\mathbf{c}}$ for each $\mathbf{P}(\vec{\mathbf{c}})$ in $\alpha(\vec{\mathbf{a}},\vec{\mathbf{b}})$.

For each $\mathbf{P}(\vec{\mathbf{c}})$ in $\alpha(\vec{\mathbf{a}},\vec{\mathbf{b}})$ we include in $\mathcal{K}'_{\alpha}$

$$\vec{\mathbf{c}} \sqsubseteq A_{\mathbf{P}}$$

and for each $A(c)$ in $\alpha(\vec{\mathbf{a}},\vec{\mathbf{b}})$ we include

$$c \sqsubseteq A.$$

For each $\vec{\mathbf{c}} = c_1,\ldots,c_n$ we include

$$\vec{\mathbf{c}} \equiv \exists f_1.c_1 \sqcap \cdots \sqcap \exists f_n.c_n \sqcap \forall f_{n+1}.\bot$$

and for each c_i we include

$$c_i \sqsubseteq \neg\exists^{\geq 2} f_i^-.\vec{\mathbf{c}}.$$

The size of $\mathcal{K}'_{\alpha}$ is polynomial in the size of q_1.

Intuitively, $\mathcal{K}'_{\alpha}$ is very close to a "naive encoding" of an ABox as a TBox. In principle, we build an ABox corresponding to $\alpha(\vec{\mathbf{a}},\vec{\mathbf{b}})$ and then we encode it as a TBox introducing the sk-concepts representing objects. The encoding is that of [7], except that: (i) we allow two objects in the ABox to denote the same individual; (ii) we force objects representing tuples to be completely determined by the objects representing their components.

Encoding of $\neg\exists\vec{\mathbf{z}}.\beta(\vec{\mathbf{a}},\vec{\mathbf{z}})$

We call *parameters* all a_i and z_j appearing in $\beta(\vec{\mathbf{a}},\vec{\mathbf{z}})$. $\mathcal{K}'_{\beta}$ encodes $\neg\exists\vec{\mathbf{z}}.\beta(\vec{\mathbf{a}},\vec{\mathbf{z}})$ by making use of the so called *tuple-graph*, which is a directed graph having:

- One node for each parameter v and one node for each $\vec{\mathbf{v}}$ such that $\mathbf{P}(\vec{\mathbf{v}})$ appears in $\neg\exists\vec{\mathbf{z}}.\beta(\vec{\mathbf{a}},\vec{\mathbf{z}})$. Each node corresponding to v is labeled by v itself and all A such that $A(v)$ appears in $\neg\exists\vec{\mathbf{z}}.\beta(\vec{\mathbf{a}},\vec{\mathbf{z}})$. Similarly, each node corresponding to $\vec{\mathbf{v}}$ is labeled by $\vec{\mathbf{v}}$ and all $\mathbf{P}$ such that $\mathbf{P}(\vec{\mathbf{v}})$ appears in $\neg\exists\vec{\mathbf{z}}.\beta(\vec{\mathbf{a}},\vec{\mathbf{z}})$.
- For each node labeled by $\vec{\mathbf{v}} = v_1,\ldots,v_n$, one edge labeled by i to the node labeled by v_i, for $i \in \{1,\ldots,n\}$.

In general the tuple-graph may be composed of $m \geq 1$ connected components. For the i-th connected component we build a concept expression $\delta_i(\vec{\mathbf{z}})$ by starting from a node corresponding to a parameter and visiting the corresponding component as follows (let u be the current node in the visit):

- If u corresponds to a parameter v, then construct the concept as the conjunction of: (i) v, if u is either a_i, or a z_j appearing in a cycle in the tuple-graph, $\top_1$ otherwise; (ii) every concept labeling the node

u; (iii) one concept $\exists f_i^-.C$ for each non-marked edge labeled by i from a node $\vec{\mathbf{v}}$ to v, where C is the concept resulting by marking the edge and visiting the node corresponding to $\vec{\mathbf{v}}$.

- If u corresponds to a tuple $\vec{\mathbf{v}} = v_1, \ldots, v_n$, let $e_1, \ldots, e_h$ be the non-marked edges from u to the nodes corresponding to the components v_i. We mark $e_1, \ldots, e_h$ and construct the concept as the conjunction of: (i) every relation labeling the node u; (ii) one concept $\exists f_i.C$ for each edge e_j, where i is the label of e_j, and C is the concept resulting by visiting the node corresponding to v_i.
- If u has already been visited then it must correspond to a parameter v and the resulting concept is v.

Observe, that $\delta_i(\vec{\mathbf{z}})$ contains newly introduced atomic concepts z_j, one for each variable occurring in a cycle in the tuple-graph.

$\mathcal{K}'_\beta$ consists of all assertions of the form (U abbreviates $(f_1 \sqcup \cdots \sqcup f_{n_{max}} \sqcup f_1^- \sqcup \cdots \sqcup f_{n_{max}}^-)^*$)

$$\top \ \sqsubseteq \ \neg(\exists U.\delta_1(\vec{\mathbf{z}}) \sqcap \cdots \sqcap \exists U.\delta_m(\vec{\mathbf{z}}))$$

obtained by replacing each z_j occurring in the assertion above by each sk-concept corresponding to a constant in $\alpha(\vec{\mathbf{a}}, \vec{\mathbf{b}})$. Observe that the number of assertions in $\mathcal{K}_\beta$ is $O(\ell_2^{\ell_1})$, where ℓ_1 is the number of variables in q_1 and ℓ_2 is the number of variables z_j occurring in a cycle in the tuple-graph.

The special attention that $\mathcal{K}'_\beta$ needs is due to the fact that the variables $\vec{\mathbf{z}}$ in $\neg\exists\vec{\mathbf{z}}.\beta(\vec{\mathbf{a}}, \vec{\mathbf{z}})$ are universally quantified. Conceptually we need to distinguish between two cases, depending on whether there is a cycle in the *tuple-graph* for q_2. The tuple-graph for a query reflects the dependency between variables and tuples resulting from the appearance of the variables in the atoms:

- If there is no cycle in the tuple-graph then we can directly build a concept expressing the constraints in $\neg\exists\vec{\mathbf{z}}.\beta(\vec{\mathbf{a}}, \vec{\mathbf{z}})$.
- If there is a cycle then, due to the fundamental inability of expressing in DLs that two chains of links meet the same individual, no concept can directly express the constraints in $\neg\exists\vec{\mathbf{z}}.\beta(\vec{\mathbf{a}}, \vec{\mathbf{z}})$. For the same reason, however, the only cycles that can be enforced in the models are those formed by the objects corresponding to the constants $\vec{\mathbf{a}}, \vec{\mathbf{b}}$. Therefore it suffices to build a concept for each possible instantiation for the variables z_i, appearing in some cycle of the tuple-graph, with the objects corresponding to $\vec{\mathbf{a}}, \vec{\mathbf{b}}$.

Encoding of objects

$\mathcal{K}'_{aux}$ enforces that for each sk-concept representing a constant in $\alpha(\vec{\mathbf{a}}, \vec{\mathbf{b}})$, a specific instance representative of the constant can be singled out. This is done, similarly to [7] by one assertion of the form

$$(\vec{\mathbf{c}} \sqcap C) \ \sqsubseteq \ \forall U.(\neg\vec{\mathbf{c}} \sqcup C)$$
$$(c \sqcap C) \ \sqsubseteq \ \forall U.(\neg c \sqcup C)$$

for each sk-concept $\vec{\mathbf{c}}$, c in $\mathcal{K}'_\alpha$, and each concept C satisfying the conditions specified in [7]. The number of such assertions is polynomial in the size of $\mathcal{K}'_\mathcal{K} \cup \mathcal{K}'_\alpha \cup \mathcal{K}'_\beta$.

The concept C'

Let $s_1, \ldots, s_t$ be all sk-concepts. Then

$$C' \ = \ \exists \mathit{create}.s_1 \sqcap \cdots \sqcap \exists \mathit{create}.s_t$$

where *create* is a newly introduced atomic role [7]. Thus C' expresses the existence of an instance for each of the sk-concepts.

Theorem 1 *$\mathit{fol}(\mathcal{K}) \wedge \alpha(\vec{\mathbf{a}}, \vec{\mathbf{b}}) \wedge \neg\exists\vec{\mathbf{z}}.\beta(\vec{\mathbf{a}}, \vec{\mathbf{z}})$ is unsatisfiable iff C' is unsatisfiable in $\mathcal{K}'$.*

Since satisfiability of a concept in a $\mathcal{CIQ}$ knowledge base is an EXPTIME complete problem [6], we get the following upper bound for conjunctive query containment.

Theorem 2 *Deciding whether a conjunctive query q_1 is contained in a conjunctive query q_2 wrt to a $\mathcal{DLR}$ knowledge base $\mathcal{K}$ can be done in time $O(2^{p(|\mathcal{K}|\cdot\ell_2^{\ell_1})})$, where $|\mathcal{K}|$ is the size of $\mathcal{K}$, ℓ_1 is the number of variables in q_1, and ℓ_2 is the number of existentially quantified variables in q_2 that appear in a cycle of the tuple-graph for q_2.*

4 Discussion

The DL $\mathcal{DLR}$ has been designed with the primary goal of giving n-ary relations the status of first-order citizen in the language. Looking at the constructs of $\mathcal{DLR}$, one realizes that the typical way of "traversing" roles in DLs by $\exists$ and $\forall$ is extended in order to "traverse" n-ary relations. The extension is achieved through the possibility of denoting the components of an n-ary relation (by means of $\$i$).

Our logic allows for inclusion assertions not only on concepts, but also on relations. Relations in turn can be either atomic or complex. This is a distinguishing feature of our approach compared to any other DL we know of. We believe that the possibility of expressing general inclusion assertions is crucial in applications such as data integration. For example, certain forms of interschema assertions [4] can be expressed only by means of inclusion assertions on complex relations.

As we said in the introduction, $\mathcal{DLR}$ is able to capture a great variety of data models and knowledge representation formalisms. For example, from the language of $\mathcal{DLR}$ one obtains:

- the relational model, by considering only atomic relations and atomic concepts;
- the entity relationship model in a straightforward way [3];
- an object-oriented data model, by restricting the use of existential and universal quantifications in concept expressions, by restricting the attention to binary relations, and by eliminating negation and disjunction;

- a "more traditional" DL, by restricting the attention to binary relations.

The problem of conjunctive query containment with respect to a DL knowledge base was also addressed in the setting of CARIN [9]. Our proposal extends that work in two main aspects. The most obvious difference is that assertions on n-ary relations are allowed in the knowledge base, as discussed above. Additionally, $\mathcal{DLR}$ enables to specify the traversal of relations in arbitrary order, in particular to specify the inverse of binary relations. Thus, also restricting $\mathcal{DLR}$ to only binary relations, we get a DL which is strictly more expressive than the DL $\mathcal{ALCNR}$ used in CARIN. Observe that the possibility of expressing inverse roles and number restrictions causes our logic to lose the finite model property, making approaches to reasoning based on model construction, such as constraint systems, not directly applicable.

The result presented in this paper can be extended to even more expressive DLs. Indeed, our method for query containment still works if we add an ABox to the knowledge base, and if we consider queries with constants. We are presently working on extending the query containment algorithm to the language $\mathcal{CVL}$ [3], which includes records, sets, and non first order features such as transitive closure and well foundedness.

Acknowledgments

This research was partially supported by Esprit LTR Project No. 22469 - Foundations of Data Warehouse Quality (DWQ).

References

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

[2] Martin Buchheit, Francesco M. Donini, Werner Nutt, and Andrea Schaerf. Terminological systems revisited: Terminology = schema + views. In *Proc. of the 12th Nat. Conf. on Artificial Intelligence (AAAI-94)*, pages 199–204, Seattle, USA, 1994.

[3] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Structured objects: Modeling and reasoning. In *Proc. of the 4th Int. Conf. on Deductive and Object-Oriented Databases (DOOD-95)*, number 1013 in Lecture Notes in Computer Science, pages 229–246. Springer-Verlag, 1995.

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

[5] Giuseppe De Giacomo and Maurizio Lenzerini. Description logics with inverse roles, functional restrictions, and n-ary relations. In *Proc. of the 4th European Workshop on Logics in Artificial Intelligence (JELIA-94)*, volume 838 of *Lecture Notes in Artificial Intelligence*, pages 332–346. Springer-Verlag, 1994.

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

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

[8] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Andrea Schaerf. A hybrid system integrating Datalog and concept languages. In *Proc. of the 2nd Conf. of the Italian Association for Artificial Intelligence (AI*IA-91)*, number 549 in Lecture Notes in Artificial Intelligence. Springer-Verlag, 1991. An extended version appeared also in the Working Notes of the AAAI Fall Symposium "Principles of Hybrid Reasoning".

[9] Alon Y. Levy and Marie-Christine Rousset. CARIN: A representation language combining Horn rules and description logics. In *Proc. of the 12th European Conf. on Artificial Intelligence (ECAI-96)*, pages 323–327, 1996.

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

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