Conjunctive Query Containment and Answering under Description Logic Constraints

DIEGO CALVANESE
Free University of Bozen-Bolzano

GIUSEPPE DE GIACOMO and MAURIZIO LENZERINI
Università di Roma "La Sapienza"

Query containment and query answering are two important computational tasks in databases. While query answering amounts to compute the result of a query over a database, query containment is the problem of checking whether for every database, the result of one query is a subset of the result of another query.

In this paper, we deal with unions of conjunctive queries, and we address query containment and query answering under Description Logic constraints. Every such constraint is essentially an inclusion dependency between concepts and relations, and their expressive power is due to the possibility of using complex expressions in the specification of the dependencies, e.g., intersection and difference of relations, special forms of quantification, regular expressions over binary relations. These types of constraints capture a great variety of data models, including the relational, the entity-relationship, and the object-oriented model, all extended with various forms of constraints. They also capture the basic features of the ontology languages used in the context of the Semantic Web.

We present the following results on both query containment and query answering. We provide a method for query containment under Description Logic constraints, thus showing that the problem is decidable, and analyze its computational complexity. We prove that query containment is undecidable in the case where we allow inequalities in the right-hand side query, even for very simple constraints and queries. We show that query answering under Description Logic constraints can be reduced to query containment, and illustrate how such a reduction provides upper bound results with respect to both combined and data complexity.

Categories and Subject Descriptors: I.2.4 [**Knowledge Representation Formalisms and Methods**]: Representation languages; F.4.1 [**Mathematical Logic**]: Computational logic, Modal logic; H.2.3 [**Languages**]: Query languages

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Description logics, Conjunctve queries, Query containment, Computational compexity

Authors' address: D. Calvanese, Faculty of Computer Science, Free University of Bozen-Bolzano, Piazza Domenicani 3, 39100 Bolzano, Italy, email: `calvanese@inf.unibz.it`; G. De Giacomo and M. Lenzerini, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", Via Salaria 113, 00198 Roma, Italy, email: `degiacomo,lenzerini@dis.uniroma1.it`.

This paper is an extended and revised version of a paper published in the Proceedings of the 17th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS'98).

1. INTRODUCTION

Query containment and query answering are two important computational tasks in databases. While query answering amounts to compute the result of a query over a database, query containment is the problem of checking whether for every database, the result of one query is a subset of the result of another query[1]. Many papers point out that checking containment is a relevant task in several contexts, including information integration [Ullman 1997], query optimization [Abiteboul et al. 1995; Aho et al. 1979a], (materialized) view maintenance [Gupta and Mumick 1995], data warehousing [Widom (ed.) 1995], constraint checking [Gupta et al. 1994], and semantic caching [Amir et al. 2003].

In this paper, we deal with query containment and query answering under integrity constraints, or simply constraints.

The former is the problem of checking whether containment between two queries holds for every database satisfying a given set of constraints. This problem arises in those situation where one wants to check query containment relatively to a database schema specified with a rich data definition language. For example, in the case of information integration, queries are often to be compared relatively to (inter-schema) constraints, which are used to declaratively specify the "glue" between two source schemas, and between one source schema and the global schema [Calvanese et al. 1998; Hull 1997; Ullman 1997; Catarci and Lenzerini 1993; Levy et al. 1995; Lenzerini 2002; Halevy 2001].

The complexity of query containment in the absence of constraints has been studied in various settings. In [Chandra and Merlin 1977], NP-completeness has been established for conjunctive queries, and in [Chekuri and Rajaraman 1997] a multi-parameter analysis has been performed for the same case, showing that the intractability is due to certain types of cycles in the queries. In [Klug 1988; van der Meyden 1998], Π_2^p-completeness of containment of conjunctive queries with inequalities was proved, and in [Sagiv and Yannakakis 1980] the case of queries with the union and difference operators was studied. For various classes of Datalog queries with inequalities, decidability and undecidability results were presented in [Chaudhuri and Vardi 1992; van der Meyden 1998; Bonatti 2004; Calvanese et al. 2003], respectively.

Query containment under constraints has also been the subject of several investigations. For example, decidability of conjunctive query containment was investigated in [Aho et al. 1979b] under functional and multi-valued dependencies, in [Johnson and Klug 1984] under functional and inclusion dependencies, in [Chan 1992; Levy and Rousset 1996; Levy and Suciu 1997] under constraints representing *is-a* hierarchies and complex objects, and in [Dong and Su 1996] in the case of constraints represented as Datalog programs. Undecidability is proved in [Calvanese and Rosati 2003] for recursive queries under inclusion dependencies. Several results on containment of XML queries under constraints expressed as DTDs are reported in [Neven and Schwentick 2003; Wood 2003].

Query answering under constraints is the problem of computing the answers to a

[1]We refer to the set semantics of query containment. Bag semantics is studied, for example, in [Ioannidis and Ramakrishnan 1995].

query over an incomplete database relatively to a set of constraints [van der Meyden 1998]. Since an incomplete database is partially specified, this task amounts to compute the tuples that satisfy the query in every database that conforms to the partial specification, and satisfies the constraints. It is well known in the database literature that there is a tight connection between the problems of conjunctive query containment and conjunctive query answering [Chandra and Merlin 1977]. Since this relationship holds also in the presence of constraints, most of the results reported above apply to query answering as well. In this paper, we concentrate mainly on query containment, and address query answering only in Section 5.

In this paper[2], we address query containment and answering in a setting where:

— The schema is constituted by concepts (unary relations) and relations as basic elements, and by a set of constraints expressed in a variant of Description Logics [Baader et al. 2003]. Every constraint is an inclusion of the form $\alpha_1 \subseteq \alpha_2$, where α_1 and α_2 are complex expressions built by using intersection and difference of relations, special forms of quantification, regular expressions over binary relation, and number restrictions (i.e., cardinality constraints imposing limitations on the number of tuples in a certain relation in which an object may appear). The constraints express essentially inclusion dependencies between concepts and relations, and their expressive power is due to the possibility of using complex expressions in the specification of the dependencies. It can be shown that our formalism is able to capture a great variety of data models, including the relational, the entity-relationship, and the object-oriented model, all extended with various forms of constraints. The relevance of the constraints dealt with in this paper is also testified by the large interest that the Semantic Web community expresses towards Description Logics. Indeed, several papers point out that ontologies play a key role in developing Semantic Web tools [Gruber 1993], and Description Logics are regarded as the main formalisms for the specification of ontologies in this context [Patel-Schneider et al. 2004]. The results presented in this paper are also relevant as a formal analysis on querying ontologies. In particular, $\mathcal{DLR}_{reg}$ can be considered as one of the most expressive Description Logics studied in the literature [Calvanese and De Giacomo 2003]

— Queries are formed as disjunctions of conjunctive queries whose atoms are concepts and relations, and therefore can express non-recursive Datalog programs. Regular expressions are confined to concepts and relations, and do not constitute atoms per se.

— An incomplete database is specified as a set of facts asserting that a specific object is an instance of a concept, or that a specific tuple of objects is an instance of a relation. As we said before, an incomplete database $\mathcal{D}$ is intended to provide a partial specification of a database, in the sense that a database conforming to $\mathcal{D}$ contains all facts explicitely asserted in $\mathcal{D}$, and may contain additional intances of concepts and relations.

We observe that, given the form of constraints and queries allowed in

[2]This paper is an improved and extended version of part of [Calvanese et al. 1998]. In that paper we overlooked a technical detail in the encoding of the containment problem, which was pointed out in [Horrocks et al. 2000]. Here we present a corrected version of the original encoding.

our approach, none of the previous results can be applied to get decidability/undecidability of query containment and query answering in our setting.

We present the following results on both query containment and query answering:

(1) We provide a method for query containment under Description Logic constraints, thus showing that the problem is decidable, and analyze its computational complexity. This result is obtained by resorting to a translation of the schema and the containment to be checked into a particular Propositional Dynamic Logic (PDL) formula, and then verifying the unsatisfiability of the formula. The technique is justified by the fact that reasoning about the schema itself (without considering the queries) is optimally done within the framework of PDL [De Giacomo and Lenzerini 1996].
(2) We prove that query containment is undecidable in the case where we allow inequalities in the right-hand side query, even for very simple constraints and queries.
(3) We show that query answering under Description Logic constraints can be reduced to query containment, and illustrate how such a reduction provides upper bound results with respect to both combined and data complexity.

The paper is organized as follows. In Section 2, we present the formalism used to express both the constraints in the schema, and the queries. In Section 3, we deal with query containment. In particular, in Subsection 3.1 we describe the logic CPDL$_g$, which will be used for deciding query containment, in Subsection 3.2 we describe the reduction of query containment to unsatisfiability in CPDL$_g$, in Subsection 3.3 we prove its correctness, and in Section 3.4 we analyze the complexity bounds for checking containment of queries. In Section 4, we show undecidability of query containment in the presence of inequalities. In Section 5, we deal with query answering, and in Section 6 we conclude the paper.

2. SCHEMAS AND QUERIES IN $\mathcal{DLR}_{reg}$

To specify database schemas and queries, we use the logical language $\mathcal{DLR}_{reg}$, inspired by [Catarci and Lenzerini 1993; Calvanese et al. 1995], belonging to the family of (expressive) Description Logics [Calvanese et al. 2001; Baader et al. 2003]. The language is based on the relational model, in the sense that a schema $\mathcal{S}$ describes the properties of a set of relations, while a query for $\mathcal{S}$ denotes a relation that is supposed to be computed from any database conforming to $\mathcal{S}$. A schema is specified in terms of a set of assertions on relations, which express the constraints that must be satisfied by every conforming database. Note that such a notion of schema corresponds to that of TBox in Description Logics [Baader et al. 2003].

2.1 Schemas

The basic elements of $\mathcal{DLR}_{reg}$ are *concepts* (unary relations), *n-ary relations*, and *regular expressions* built over projections of relations on two of their components[3].

We assume to deal with a finite set of atomic concepts and relations, denoted by A and $\mathbf{P}$ respectively. Each atomic relation has an associated *arity*, which is an

[3]We could include in the logic also domains, i.e. sets of values such as integer, string, etc.. However, for the sake of simplicity, we do not consider this aspect in this work.

$$
\begin{array}{rcl}
\top_1^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \\
A^{\mathcal{I}} & \subseteq & \Delta^{\mathcal{I}} \\
(\neg C)^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
(C_1 \sqcap C_2)^{\mathcal{I}} & = & C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}} \\
(\exists E.C)^{\mathcal{I}} & = & \{d \in \Delta^{\mathcal{I}} \mid \exists d' \in C^{\mathcal{I}}.(d,d') \in E^{\mathcal{I}}\} \\
(\exists [\$i]\mathbf{R})^{\mathcal{I}} & = & \{d \in \Delta^{\mathcal{I}} \mid \exists (d_1,\ldots,d_n) \in \mathbf{R}^{\mathcal{I}}.d_i = d\} \\
(\leq k\,[\$i]\mathbf{R})^{\mathcal{I}} & = & \{d \in \Delta^{\mathcal{I}} \mid \#\{(d_1,\ldots,d_n) \in \mathbf{R}_1^{\mathcal{I}} \mid d_i = d\} \leq k\} \\
 & & \\
\top_n^{\mathcal{I}} & \subseteq & (\Delta^{\mathcal{I}})^n \\
\mathbf{P}^{\mathcal{I}} & \subseteq & \top_n^{\mathcal{I}} \\
(\$i/n\!:\!C)^{\mathcal{I}} & = & \{(d_1,\ldots,d_n) \in \top_n^{\mathcal{I}} \mid d_i \in C^{\mathcal{I}}\} \\
(\neg \mathbf{R})^{\mathcal{I}} & = & \top_n^{\mathcal{I}} \setminus \mathbf{R}^{\mathcal{I}} \\
(\mathbf{R}_1 \sqcap \mathbf{R}_2)^{\mathcal{I}} & = & \mathbf{R}_1^{\mathcal{I}} \cap \mathbf{R}_2^{\mathcal{I}} \\
 & & \\
(\mathbf{R}|_{\$i,\$j})^{\mathcal{I}} & = & \{(x_i,x_j) \mid (x_1,\ldots,x_n) \in \mathbf{R}^{\mathcal{I}}\} \\
\varepsilon^{\mathcal{I}} & = & \{(x,x) \mid x \in \Delta^{\mathcal{I}}\} \\
(E_1 \circ E_2)^{\mathcal{I}} & = & E_1^{\mathcal{I}} \circ E_2^{\mathcal{I}} \\
(E_1 \sqcup E_2)^{\mathcal{I}} & = & E_1^{\mathcal{I}} \cup E_2^{\mathcal{I}} \\
(E^*)^{\mathcal{I}} & = & (E^{\mathcal{I}})^* \\
\end{array}
$$

Fig. 1. Semantic rules for $\mathcal{DLR}_{reg}$ ($\mathbf{P}$, $\mathbf{R}$, $\mathbf{R}_1$, and $\mathbf{R}_2$ have arity n)

integer greater than 1. We use C to denote arbitrary concepts, $\mathbf{R}$ to denote arbitrary relations, and E to denote regular expressions, respectively built according to the following syntax

$$
\begin{array}{l}
C ::= \top_1 \mid A \mid \neg C \mid C_1 \sqcap C_2 \mid \exists E.C \mid \exists [\$i]\mathbf{R} \mid (\leq k\,[\$i]\mathbf{R}) \\
\mathbf{R} ::= \top_n \mid \mathbf{P} \mid (\$i/n\!:\!C) \mid \neg \mathbf{R} \mid \mathbf{R}_1 \sqcap \mathbf{R}_2 \\
E ::= \mathbf{R}|_{\$i,\$j} \mid \varepsilon \mid E_1 \circ E_2 \mid E_1 \sqcup E_2 \mid E^*
\end{array}
$$

where i, j are positive integers that intuitively denote components of relations, n is a positive integer greater than 1 denoting the arity of a relation, and k is a nonnegative integer. Actually, we restrict the attention to concepts and relations that are *well-typed*, which means that

—only relations of the same arity n are combined to form expressions of type $\mathbf{R}_1 \sqcap \mathbf{R}_2$ (which inherit the arity n), and

—$i \leq n$ whenever i denotes a component of a relation of arity n.

A $\mathcal{DLR}_{reg}$ *schema* is constituted by a finite set of *assertions*, of the form

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

where $\mathbf{R}_1$ and $\mathbf{R}_2$ are of the same arity. Note that our notion of schema corresponds to that of TBox in Description Logics [Baader et al. 2003].

The semantics of $\mathcal{DLR}_{reg}$ is specified through the notion of interpretation. An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ of a $\mathcal{DLR}_{reg}$ schema $\mathcal{S}$ and a set $\mathcal{C}$ (of constants to be used in queries) is constituted by an *interpretation domain* $\Delta^{\mathcal{I}}$ and an *interpretation function* $\cdot^{\mathcal{I}}$ that assigns:

—to each constant c in $\mathcal{C}$, an element $c^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$ under the *unique name assumption*, i.e., for any two distinct constants c_1 and c_2 of $\Delta^{\mathcal{I}}$, we have that $c_1^{\mathcal{I}} \neq c_2^{\mathcal{I}}$;

—to each atomic concept A, a subset $A^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$;

—to each relation $\mathbf{P}$ of arity n, a subset $\mathbf{P}^{\mathcal{I}}$ of $(\Delta^{\mathcal{I}})^n$;

The interpretation function is extended to arbitrary concepts, arbitrary relations, and regular expressions in such a way that the conditions in Figure 1 are satisfied.

We observe that $\top_1$ denotes the interpretation domain, while $\top_n$, for $n > 1$, does *not* denote the n-Cartesian product of the domain, but only a subset of it, that covers all relations of arity n. It follows from this property that the "$\neg$" constructor on relations is used to express difference of relations, rather than complement. The constructors "$\neg$" and "$\sqcap$" on concepts have the usual meaning of negation and conjunction, respectively. The expression $\exists[\$i]\mathbf{R}$ denotes the projection of relation $\mathbf{R}$ on its i-th component. An expression of the form $(\leq k\,[\$i]\mathbf{R})$ is called a *number restriction*, and imposes a limit on the number of times an object can participate to relation $\mathbf{R}$ as i-th component. The expression $(\$i/n:C)$ represents those tuples of arity n whose i-th component is an instance of concept C. Finally, E are regular expressions build over projections of relations on two of their components.

In what follows, we abbreviate $\neg\exists E.\neg C$ with $\forall E.C$, and $(\$i/n:C)$ with $(\$i:C)$ when n is clear from the context.

An interpretation $\mathcal{I}$ *satisfies* an assertion $C_1 \sqsubseteq C_2$ (resp., $\mathbf{R}_1 \sqsubseteq \mathbf{R}_2$) if $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$ (resp., $\mathbf{R}_1^{\mathcal{I}} \subseteq \mathbf{R}_2^{\mathcal{I}}$). An interpretation that satisfies all assertions in a schema $\mathcal{S}$ is called a *model* of $\mathcal{S}$. It is easy to see that a model of a schema $\mathcal{S}$ actually corresponds to a database conforming to S, i.e., a database satisfying all the constraints represented by $\mathcal{S}$. A schema is *satisfiable* if it admits a model. A schema $\mathcal{S}$ *logically implies* an inclusion assertion $C_1 \sqsubseteq C_2$ (resp. $\mathbf{R}_1 \sqsubseteq \mathbf{R}_2$) if for every model $\mathcal{I}$ of $\mathcal{S}$ we have that $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$ (resp. $\mathbf{R}_1^{\mathcal{I}} \subseteq \mathbf{R}_2^{\mathcal{I}}$).

It can be shown that $\mathcal{DLR}_{reg}$ is able to capture a great variety of data models with many forms of constraints. For example, we obtain the entity-relationship model (including *is-a* relations on both entities and relations) in a straightforward way [Calvanese et al. 1995], and an object-oriented data model (extended with several types of constraints), by restricting the use of existential and universal quantifications in concept expressions, by restricting the attention to binary relations, and by eliminating negation, disjunction and regular expressions. Compared with the relational model, the following observations point out the kinds of constraints that can be expressed using $\mathcal{DLR}_{reg}$.

—Assertions directly express a special case of typed inclusion dependencies, namely the one where no projection of relations is used.

—Unary inclusion dependencies are easily expressible by means of the $\exists[\$2]\mathbf{P}$ construct. For example, $\exists[\$2]\mathbf{P}_1 \sqsubseteq \exists[\$3]\mathbf{P}_2$ is a unary inclusion dependency between attribute 2 of $\mathbf{P}_1$ and attribute 3 of $\mathbf{P}_2$.

—Existence and exclusion dependencies are expressible by means of $\exists$ and $\neg$, respectively.

—A limited form of functional dependencies can be expressed by means of $(\leq 1\,[\$i]\mathbf{R})$. For example, $\top_1 \sqsubseteq (\leq 1\,[\$i]\mathbf{P})$ specifies that attribute i functionally determines all other attributes of $\mathbf{P}$.

—The possibility of constructing complex expressions provides a special form of view definition. Indeed, the two assertions $\mathbf{P} \sqsubseteq \mathbf{R}$, $\mathbf{R} \sqsubseteq \mathbf{P}$ (where $\mathbf{R}$ is a complex expression) is a view definition for $\mathbf{P}$. Notably, views can be freely used in assertions (even with cyclic references), and, therefore, all the above discussed constraints can be imposed not only on atomic relations, but also on views. These features make our logic particularly suited for expressing inter-schema relationships in the context of information integration [Calvanese et al. 1998], where it is crucial to be able to state that a certain concept of a schema corresponds (by means of inclusion or equivalence) to a view in another schema.

—Finally, regular expressions can be profitably used to represent in the schema inductively defined structures such as sequences and lists, imposing complex conditions on them.

One of the distinguishing features of $\mathcal{DLR}_{reg}$ is that it is equipped with a method for checking logical implication. Indeed, $\mathcal{DLR}_{reg}$ shares ExpTime-completeness of schema satisfiability and logical implication with many expressive Description Logics [Calvanese et al. 2001; Baader et al. 2003] (see below).

We point out that $\mathcal{DLR}_{reg}$ supports only special forms of functional and inclusion dependencies. Hence the undecidability result of implication for (general) functional and inclusion dependencies taken together, shown in [Mitchell 1983; Chandra and Vardi 1985], does not apply.

We finally observe that $\mathcal{DLR}_{reg}$ does not enjoy the *finite model property*, similarly to all Description Logics that include the ability to predicate at least on binary relations, stating their functionality and navigating them in both directions [Calvanese and De Giacomo 2003].

2.2 Queries

A query q for a $\mathcal{DLR}_{reg}$ schema is a union of conjunctive queries, written (using Datalog notation) in the form:

$$q(\vec{\mathbf{x}}) \leftarrow conj_1(\vec{\mathbf{x}}, \vec{\mathbf{y}}_1, \vec{\mathbf{c}}_1) \vee \cdots \vee conj_m(\vec{\mathbf{x}}, \vec{\mathbf{y}}_m, \vec{\mathbf{c}}_m)$$

where each $conj_i(\vec{\mathbf{x}}, \vec{\mathbf{y}}_i, \vec{\mathbf{c}}_i)$ is a conjunction of *atoms*, and $\vec{\mathbf{x}}, \vec{\mathbf{y}}_i$ (resp. $\vec{\mathbf{c}}_i$) are exactly the variables (resp. constants) appearing in the i-th conjunction[4]. Each atom has one of the forms $C(t)$ or $\mathbf{R}(\vec{\mathbf{t}})$, where

—t and $\vec{\mathbf{t}}$ are constants or variables in $\vec{\mathbf{x}}, \vec{\mathbf{y}}_i, \vec{\mathbf{c}}_i$

—C and $\mathbf{R}$ are respectively concepts and relations expressions over $\mathcal{S}$.

The number of variables of $\vec{\mathbf{x}}$ is called the *arity* of q, i.e., the arity of the relation denoted by the query q.

We observe that the atoms in the queries are arbitrary $\mathcal{DLR}_{reg}$ concepts and relations, freely used in the assertions of the schema. This distinguishes our approach with respect to [Donini et al. 1998; Levy and Rousset 1996], where no constraints can be expressed in the schema on the relations that appear in the queries. Note

[4]Note that the assumption that the variables $\vec{\mathbf{x}}$ appear in all conjunctions is not a limitation, since we can always add to a conjunction new atoms $\top_1(x)$, one for each variable x that does not originally appear in the conjunction.

also that regular expressions may appear in concepts and relations. However, they cannot be used to form binary atoms specifying that two variables or constants are connected by a path described by the regular expression [Calvanese et al. 2000].

Given an interpretation $\mathcal{I}$ of a schema $\mathcal{S}$, a query q for $\mathcal{S}$ of arity n is interpreted as the set $q^{\mathcal{I}}$ of n-tuples $(o_1, \ldots, o_n)$, with each $o_i \in \Delta^{\mathcal{I}}$, such that, under the assignment of o_i to x_i, for $i \in \{1, \ldots, n\}$, the formula

$$\exists \vec{\mathbf{y}}_1 . \mathit{conj}_1(\vec{\mathbf{x}}, \vec{\mathbf{y}}_1, \vec{\mathbf{c}}_1) \vee \cdots \vee \exists \vec{\mathbf{y}}_m . \mathit{conj}_m(\vec{\mathbf{x}}, \vec{\mathbf{y}}_m, \vec{\mathbf{c}}_m)$$

evaluates to true in $\mathcal{I}$ [Enderton 1972].

If q and q' are two queries (of the same arity) for $\mathcal{S}$, we say that q is *contained in q' with respect to $\mathcal{S}$*, denoted $\mathcal{S} \models q \subseteq q'$, if $q^{\mathcal{I}} \subseteq q'^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{S}$. The query q is *satisfiable with respect to $\mathcal{S}$*, denoted $\mathcal{S} \not\models q \equiv \emptyset$, if there is a model $\mathcal{I}$ of $\mathcal{S}$ such that $q^{\mathcal{I}} \neq \emptyset$.

Query containment (with respect to a schema) is the problem of checking whether $\mathcal{S} \models q \subseteq q'$, where $\mathcal{S}$, q, and q' are given as input. *Query satisfiability (with respect to a schema)* is the problem of checking whether $\mathcal{S} \not\models q \equiv \emptyset$, where $\mathcal{S}$ and q are given as input.

We provide now an alternative characterization of query containment, on which we base the technique presented in Section 3. Given a $\mathcal{DLR}_{reg}$ schema $\mathcal{S}$ and two queries for $\mathcal{S}$

$$\begin{aligned} q(\vec{\mathbf{x}}) &\leftarrow \mathit{conj}_1(\vec{\mathbf{x}}, \vec{\mathbf{y}}_1, \vec{\mathbf{c}}_1) \vee \cdots \vee \mathit{conj}_m(\vec{\mathbf{x}}, \vec{\mathbf{y}}_m, \vec{\mathbf{c}}_m) \\ q'(\vec{\mathbf{x}}) &\leftarrow \mathit{conj}'_1(\vec{\mathbf{x}}, \vec{\mathbf{y}}'_1, \vec{\mathbf{c}}'_1) \vee \cdots \vee \mathit{conj}'_{m'}(\vec{\mathbf{x}}, \vec{\mathbf{y}}'_{m'}, \vec{\mathbf{c}}'_{m'}) \end{aligned}$$

we have that $\mathcal{S} \models q \subseteq q'$ iff there is no model $\mathcal{I}$ of $\mathcal{S}$ such that, when assigning suitable objects in $\Delta^{\mathcal{I}}$ to $\vec{\mathbf{x}}, \vec{\mathbf{y}}_1, \ldots \vec{\mathbf{y}}_m$, the formula

$$\begin{aligned} &(\mathit{conj}_1(\vec{\mathbf{x}}, \vec{\mathbf{y}}_1, \vec{\mathbf{c}}_1) \vee \cdots \vee \mathit{conj}_m(\vec{\mathbf{x}}, \vec{\mathbf{y}}_m, \vec{\mathbf{c}}_m)) \wedge \\ &\neg \exists \vec{\mathbf{z}}_1 . \mathit{conj}'_1(\vec{\mathbf{x}}, \vec{\mathbf{z}}_1, \vec{\mathbf{c}}'_1) \wedge \cdots \wedge \neg \exists \vec{\mathbf{z}}_{m'} . \mathit{conj}'_{m'}(\vec{\mathbf{x}}, \vec{\mathbf{z}}_{m'}, \vec{\mathbf{c}}'_{m'}) \end{aligned}$$

evaluates to true in $\mathcal{I}$. In other words, $\mathcal{S} \models q \subseteq q'$ if and only if there is no model of $\mathcal{S}$ that makes the formula

$$\begin{aligned} &(\mathit{conj}_1(\vec{\mathbf{a}}, \vec{\mathbf{b}}_1, \vec{\mathbf{c}}_1) \vee \cdots \vee \mathit{conj}_m(\vec{\mathbf{a}}, \vec{\mathbf{b}}_m, \vec{\mathbf{c}}_m)) \wedge \\ &\neg \exists \vec{\mathbf{z}}_1 . \mathit{conj}'_1(\vec{\mathbf{a}}, \vec{\mathbf{z}}_1, \vec{\mathbf{c}}'_1) \wedge \cdots \wedge \neg \exists \vec{\mathbf{z}}_{m'} . \mathit{conj}'_{m'}(\vec{\mathbf{a}}, \vec{\mathbf{z}}_{m'}, \vec{\mathbf{c}}'_{m'}) \end{aligned}$$

true, where $\vec{\mathbf{a}}$, $\vec{\mathbf{b}}_1, \ldots, \vec{\mathbf{b}}_m$ are Skolem constants, i.e., constants not appearing elsewhere for which the unique name assumption does not hold.

2.3 Example

Consider an application where the departments of a given company can be controlled by other departments, and sold to companies. Every department is controlled by at most one department, and by at least one main department, possibly indirectly. A main department is not controlled by any department. If a main department is sold, then all the departments controlled by it are also sold. Finally, if a department is sold, then there is a main department controlling it, either directly or indirectly, that is also sold.

The basic concepts and relations are shown in Figure 2 in the form of an entity-relationship diagram. The specification of the application in $\mathcal{DLR}_{reg}$ makes use of

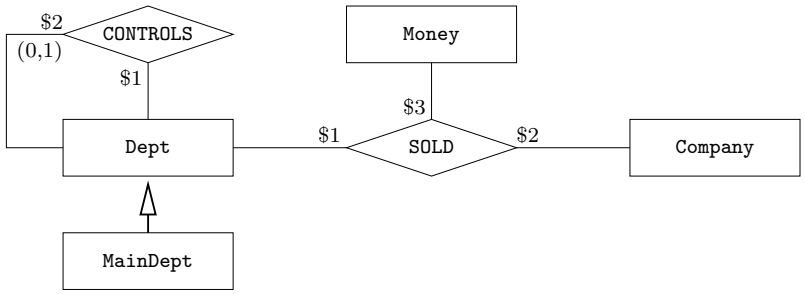


Fig. 2. The entity-relationship diagram for the example in Section 2.3

the concepts Dept, MainDept, Money, Company, and the relations CONTROLS, SOLD. In particular, $\texttt{CONTROLS}(x, y)$ means that department x has control over department y, and $\texttt{SOLD}(x, y, z)$ means that department x has been sold to company y at price z. The schema $\mathcal{S}$ is constituted by the following assertions:

$$\begin{aligned}
\texttt{SOLD} &\sqsubseteq (\$1 : \texttt{Dept}) \sqcap (\$2 : \texttt{Company}) \sqcap (\$3 : \texttt{Money}) \\
\texttt{CONTROLS} &\sqsubseteq (\$1 : \texttt{Dept}) \sqcap (\$2 : \texttt{Dept}) \\
\texttt{Dept} &\sqsubseteq (\leq 1\ [\$2]\texttt{CONTROLS}) \sqcap \exists(\texttt{CONTROLS}|_{\$2,\$1})^*.\texttt{MainDept} \\
\texttt{MainDept} &\sqsubseteq \texttt{Dept} \sqcap \neg\exists[\$2]\texttt{CONTROLS} \\
\texttt{MainDept} \sqcap \exists[\$1]\texttt{SOLD} &\sqsubseteq \forall(\texttt{CONTROLS}|_{\$1,\$2})^*.\exists[\$1]\texttt{SOLD} \\
\texttt{Dept} \sqcap \exists[\$1]\texttt{SOLD} &\sqsubseteq \exists(\texttt{CONTROLS}|_{\$2,\$1})^*.(\texttt{MainDept} \sqcap \exists[\$1]\texttt{SOLD})
\end{aligned}$$

The first two assertions are used to specify the types of the attributes of the relations. The third and the fourth assertions specify the basic properties of Dept and MainDept. It is easy to see that such assertions imply that, in all the models of $\mathcal{S}$, the set of CONTROLS links starting from an instance m of MainDept form a tree (which we call CONTROLS-tree) with root m. The role of the transitive closure $(\texttt{CONTROLS}|_{\$2,\$1})^*$ and the number restrictions is crucial for correctly representing the above property in the schema. Finally, the last two assertions, each one stating inclusions between views, specify the company policy for selling departments. Note again the use of the transitive closure for this purpose.

We now consider two queries for the schema $\mathcal{S}$. The first query, called q is used to retrieve all departments that control two departments that have been sold to the same company and such that one of them controls the other. The second query, called q', retrieves all the departments that have been sold. The queries q and q' are defined as follows:

$$\begin{aligned}
q(x) &\leftarrow \texttt{CONTROLS}(x, y) \wedge \texttt{SOLD}(y, z, z_1) \wedge \texttt{CONTROLS}(x, w) \wedge \texttt{SOLD}(w, z, z_2) \wedge \\
&\quad\ \texttt{CONTROLS}(y, w) \\
q'(x) &\leftarrow \texttt{Dept}(x) \wedge \texttt{SOLD}(x, z, z')
\end{aligned}$$

The schema $\mathcal{S}$ imposes that (i) the CONTROLS relation is typed, so that x in q is a department; (ii) each department is controlled by at most one department; (iii) when a department is sold, there is a main department (possibly indirectly) controlling it that is also sold; (iv) when a main department is sold, then all departments it

(possibly indirectly) controls are also sold. From (i–iv) it follows that a department controlling one that is sold, is sold as well. Therefore, in every model of $\mathcal{S}$ each x satisfying q also satisfies q', and hence $\mathcal{S} \models q \subseteq q'$.

Following the same arguments, it is easy to see that the following query q'' is unsatisfiable with respect to S:

$$q''(x) \leftarrow \texttt{CONTROLS}(x, y) \wedge \texttt{SOLD}(y, z_1, z_2) \wedge \neg\texttt{SOLD}(x, w_1, w_2)$$

3. CHECKING QUERY CONTAINMENT

We address the problem of deciding, given a schema $\mathcal{S}$ and two queries q and q' of the same arity, whether $\mathcal{S} \models q \subseteq q'$. To do so, we make use of a reduction of query containment to a problem of unsatisfiability in a variant of Propositional Dynamic Logic, called CPDL$_g$.

The reduction we present is based on the combination of several techniques, some of which stem from work in the Description Logic community, and some of which are novel to the present research. First, the general idea of reducing reasoning in Description Logics to reasoning in Propositional Dynamic Logics has been widely used since [Schild 1991; De Giacomo and Lenzerini 1994] to show decidability and complexity results [Calvanese and De Giacomo 2003]. Then, we exploit *reification* (cf., construction of $\Phi_{\mathcal{S}}$ in subsection 3.2) to deal with relations of arbitrary arity in a formalism whose semantic structures are based on graphs, i.e., unary and binary relations only. Such a technique was introduced in [Catarci and Lenzerini 1993; De Giacomo and Lenzerini 1995] and is at the basis of the tight relationship between Description Logics and conceptual models for databases [Calvanese et al. 1999; Borgida and Brachman 2003]. Another technique we use is the encoding of formulae denoting single objects (here called *name-formulae*, a.k.a. nominals) within standard Propositional Dynamic Logics. These techniques were introduced in [De Giacomo and Lenzerini 1994; 1996] to deal with knowledge on single individuals in expressive Description Logics. Here, we directly build on the results in [De Giacomo and Lenzerini 1996] and use the technique to deal with what is essentially a "disjunction of ABoxes", instead of a single ABox (cf., construction of Φ_{aux} and of $\bigvee_{j=1}^{m} \Phi_{conj_j}$ in subsection 3.2). The core technique to deal with containment is a novelty of this research. It is based on realizing that we can restrict our attention to models in which the only cycles are those formed by explicitly named individuals, and hence cyclic queries can only be satisfied through these individuals (cf., construction of $\bigwedge_{j=1}^{m'} \neg\Phi_{conj'_j}$ in subsection 3.2).

In the next subsection, we introduce CPDL$_g$. Then, we present the reduction, prove its correctness, and analyze the computational complexity of the resulting containment algorithm.

3.1 The Propositional Dynamic Logic CPDL$_g$

Propositional Dynamic Logics are specific modal logics originally proposed as a formal system for reasoning about computer program schemas [Fischer and Ladner 1979]. Since then PDLs have been studied extensively and extended in several ways (see e.g., [Kozen and Tiuryn 1990] for a survey).

Here, we make use of CPDL$_g$ (studied in [De Giacomo and Lenzerini 1996] in the context of description logics), which is an extension of Converse PDL [Kozen and

$$
\begin{array}{rcl}
A^{\mathcal{M}} & \subseteq & S \\
(\neg\phi)^{\mathcal{M}} & = & S \setminus \phi^{\mathcal{M}} \\
(\phi_1 \wedge \phi_2)^{\mathcal{M}} & = & \phi_1^{\mathcal{M}} \cap \phi_2^{\mathcal{M}} \\
(\langle r \rangle \phi)^{\mathcal{M}} & = & \{s \mid \exists s'.(s,s') \in r^{\mathcal{M}} \wedge s' \in \phi^{\mathcal{M}}\} \\
([p]_{\leq k}\phi)^{\mathcal{M}} & = & \{s \mid \#\{s' \mid (s,s') \in p^{\mathcal{M}} \wedge s' \in \phi^{\mathcal{M}}\} \leq k\} \\
([p^-]_{\leq k}\phi)^{\mathcal{M}} & = & \{s \mid \#\{s' \mid (s',s) \in p^{\mathcal{M}} \wedge s' \in \phi^{\mathcal{M}}\} \leq k\}
\end{array}
$$

$$
\begin{array}{rcl}
p^{\mathcal{M}} & \subseteq & S \times S \\
\varepsilon^{\mathcal{M}} & = & \{(x,x) \mid x \in S\} \\
(r_1; r_2)^{\mathcal{M}} & = & r_1^{\mathcal{M}} \circ r_2^{\mathcal{M}} \\
(r_1 \cup r_2)^{\mathcal{M}} & = & r_1^{\mathcal{M}} \cup r_2^{\mathcal{M}} \\
(r^*)^{\mathcal{M}} & = & (r^{\mathcal{M}})^* \;=\; \bigcup_{i \geq 0} (r^{\mathcal{M}})^i \\
(\phi?)^{\mathcal{M}} & = & \{(s,s) \mid s \in \phi^{\mathcal{M}}\} \\
(r^-)^{\mathcal{M}} & = & \{(s,s') \mid (s',s) \in r^{\mathcal{M}}\}
\end{array}
$$

Fig. 3. Semantic rules for CPDL_g

Tiuryn 1990] with *graded modalities* [Fattorosi-Barnaba and De Caro 1985]. The syntax of CPDL_g is as follows (A denotes an *atomic formula*, ϕ an arbitrary *formula*, p an *atomic program*, and r an arbitrary *program*):

$$
\begin{array}{l}
\phi ::= A \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \langle r \rangle \phi \mid [p]_{\leq k}\phi \mid [p^-]_{\leq k}\phi \\
r ::= p \mid \varepsilon \mid r_1; r_2 \mid r_1 \cup r_2 \mid r^* \mid \phi? \mid r^-
\end{array}
$$

We use the standard abbreviations, namely T for true, F for false, $\vee$ for disjunction, $\Rightarrow$ for material implication, and $[r]\phi$ for $\neg\langle r \rangle \neg\phi$.

As usual for PDLs, the semantics of CPDL_g is based on *Kripke structures* $\mathcal{M} = (S, \cdot^{\mathcal{M}})$, where S is a *set of states* and $\cdot^{\mathcal{M}}$ is a mapping interpreting each atomic formula A as a subset $A^{\mathcal{M}}$ of S and each atomic program p as a binary relation $p^{\mathcal{M}}$ over S. The mapping $\cdot^{\mathcal{M}}$ is then extended to arbitrary formulae and programs in such a way that the conditions in Figure 3 are satisfied.

It can be shown that CPDL_g has typical properties of PDLs, in particular the *connected-model property* (if a formula has a model, then it has one that is connected when viewing it as a graph), the *tree-model property* (if a formula has a model, then it has one that is a tree when viewing it as an undirected graph), and *EXPTIME-completeness* of checking satisfiability of a formula (with the assumption that numbers in graded modalities are represented in unary) [De Giacomo and Lenzerini 1996; Calvanese et al. 2001; Baader et al. 2003].

In the following we make use of two notions related to CPDL_g, namely the Fischer-Ladner closure of a formula and the prefix of a program. Given a CPDL_g formula ϕ, the *Fischer-Ladner closure* $CL(\phi)$ of ϕ is defined as the smallest set C of formulae

containing ϕ and such that:

$$\begin{array}{lll}
\neg\phi' \in C & \text{implies} & \phi' \in C \\
\phi' \in C & \text{implies} & \neg\phi' \in C \quad (\text{if } \phi' \neq \neg\phi'') \\
\phi_1 \wedge \phi_2 \in C & \text{implies} & \phi_1, \phi_2 \in C \\
\langle r \rangle \phi' \in C & \text{implies} & \phi' \in C \\
[p]_{\leq k}\phi \in C & \text{implies} & \phi' \in C \\
[p^-]_{\leq k}\phi \in C & \text{implies} & \phi' \in C \\
\langle r_1; r_2 \rangle \phi' \in C & \text{implies} & \langle r_1 \rangle \langle r_2 \rangle \phi' \in C \\
\langle r_1 \cup r_2 \rangle \phi' \in C & \text{implies} & \langle r_1 \rangle \phi', \langle r_2 \rangle \phi' \in C \\
\langle r^* \rangle \phi' \in C & \text{implies} & \langle r \rangle \langle r^* \rangle \phi' \in C \\
\langle \phi'? \rangle \phi'' \in C & \text{implies} & \phi' \in C
\end{array}$$

Given a CPDL$_g$ program r, the set $Pre(r)$ of *prefixes* of r is defined as follows:

$$\begin{array}{lcl}
Pre(p) & = & \{\varepsilon, p\} \\
Pre(p^-) & = & \{\varepsilon, p^-\} \\
Pre(r_1; r_2) & = & Pre(r_1) \cup \{r_1; r_2' \mid r_2' \in Pre(r_2)\} \\
Pre(r_1 \cup r_2) & = & Pre(r_1) \cup Pre(r_2) \\
Pre(r_1^*) & = & \{r_1^*; r_1' \mid r_1' \in Pre(r_1)\} \\
Pre(\phi?) & = & \{\varepsilon, \phi?\}
\end{array}$$

3.2 Reduction of Query Containment to Unsatisfiability in CPDL$_g$

Our aim is to reduce query containment to a problem of unsatisfiability in CPDL$_g$. To this end, we construct a CPDL$_g$ formula starting from an instance of the query containment problem. More precisely, if we have to check whether there is no model of $\mathcal{S}$ that makes the formula

$$\begin{array}{l}
(conj_1(\vec{\mathbf{a}}, \vec{\mathbf{b}}_1, \vec{\mathbf{c}}_1) \vee \cdots \vee conj_m(\vec{\mathbf{a}}, \vec{\mathbf{b}}_m, \vec{\mathbf{c}}_m)) \wedge \\
\neg\exists \vec{\mathbf{z}}_1 . conj_1'(\vec{\mathbf{a}}, \vec{\mathbf{z}}_1, \vec{\mathbf{c}}_1') \wedge \cdots \wedge \neg\exists \vec{\mathbf{z}}_{m'} . conj_{m'}'(\vec{\mathbf{a}}, \vec{\mathbf{z}}_{m'}, \vec{\mathbf{c}}_{m'}')
\end{array}$$

true, where $\vec{\mathbf{a}}$, $\vec{\mathbf{b}}_1, \ldots, \vec{\mathbf{b}}_m$ are Skolem constants, we check the unsatisfiability of the CPDL$_g$ formula

$$\Phi_{S \not\models q \subseteq q'} = \Phi_{\mathcal{S}} \wedge (\bigvee_{j=1}^{m} \Phi_{conj_j}) \wedge (\bigwedge_{j=1}^{m'} \neg\Phi_{conj_j'}) \wedge \Phi_{aux},$$

constructed as described below.

$\Phi_{\mathcal{S}}$: encoding of $\mathcal{S}$

$\Phi_{\mathcal{S}}$ is the translation of $\mathcal{S}$ into a CPDL$_g$ formula, that is based on *reification* of n-ary relations, i.e., a tuple in a model of $\mathcal{S}$ is represented in a model of $\Phi_{S \not\models q \subseteq q'}$ by a state having one functional link f_i for each tuple component $\$i$. Let n_{max} be the maximum arity of relations in $\mathcal{S}$. The formula $\Phi_{\mathcal{S}}$ makes use of the mapping $\sigma(\cdot)$ from $\mathcal{DLR}_{reg}$ expressions to CPDL$_g$ formulae defined in Figure 4. The atomic formula $\top_1$ denotes those states that represent objects, while each atomic formula $\top_n$, with $n \in \{2, \ldots, n_{max}\}$, denotes those states that represent tuples of arity n. We denote with U the program $(create \cup f_1 \cup \cdots \cup f_{n_{max}} \cup create^- \cup f_1^- \cup \cdots \cup f_{n_{max}}^-)^*$, where $create, f_1, \ldots, f_{n_{max}}$ are all atomic programs used in $\Phi_{S \not\models q \subseteq q'}$. Hence, U is

$$
\begin{array}{rcl}
\sigma(\top_1) & = & \top_1 \\
\sigma(A) & = & A \\
\sigma(\neg C) & = & \top_1 \wedge \neg\sigma(C) \\
\sigma(C_1 \sqcap C_2) & = & \sigma(C_1) \wedge \sigma(C_2) \\
\sigma(\exists E.C) & = & \langle\sigma(E)\rangle\sigma(C) \\
\sigma(\exists[\$i]\mathbf{R}) & = & \langle f_i^- \rangle \sigma(\mathbf{R}) \\
\sigma((\leq k\,[\$i]\mathbf{R})) & = & [f_i^-]_{\leq k}\sigma(\mathbf{R})
\end{array}
\qquad
\begin{array}{rcl}
\sigma(\top_n) & = & \top_n \\
\sigma(\mathbf{P}) & = & \mathbf{P} \\
\sigma((i/n:C)) & = & \top_n \wedge [f_i]\sigma(C) \\
\sigma(\neg\mathbf{R}) & = & \top_n \wedge \neg\sigma(\mathbf{R}) \\
\sigma(\mathbf{R}_1 \sqcap \mathbf{R}_2) & = & \sigma(\mathbf{R}_1) \wedge \sigma(\mathbf{R}_2) \\
 & & \\
\sigma(\varepsilon) & = & \varepsilon \\
\sigma(R|_{\$i,\$j}) & = & f_i^-;\sigma(R)?;f_j \\
\sigma(E_1 \circ E_2) & = & \sigma(E_1);\sigma(E_2) \\
\sigma(E_1 \sqcup E_2) & = & \sigma(E_1) \cup \sigma(E_2) \\
\sigma(E^*) & = & \sigma(E)^*
\end{array}
$$

Fig. 4. Mapping $\sigma(\cdot)$ from $\mathcal{DLR}_{reg}$ to CPDL$_g$

the so-called *master modality* [Blackburn et al. 2001], which in our case, due to the connected-model property of CPDL$_g$, represents the universal accessibility relation. Therefore, for a given interpretation, $[U]\phi$ expresses that ϕ holds in every state, and $\langle U\rangle\phi$ expresses that ϕ holds in some state.

$\Phi_{\mathcal{S}}$ is the conjunction of the following formulae:

$$[U](\top_1 \vee \cdots \vee \top_{n_{max}}) \tag{1}$$

$$[U][f_i]_{\leq 1}\mathsf{T} \qquad \text{for each } i \in \{1,\ldots,n_{max}\} \tag{2}$$

$$[U](\top_n \equiv \langle f_1\rangle\top_1 \wedge \cdots \wedge \langle f_n\rangle\top_1 \wedge [f_{n+1}]\mathsf{F}) \qquad \text{for each } n \in \{2,\ldots,n_{max}\} \tag{3}$$

$$[U]([f_i]\mathsf{F} \Rightarrow [f_{i+1}]\mathsf{F}) \qquad \text{for each } i \in \{1,\ldots,n_{max}\} \tag{4}$$

$$[U](A \Rightarrow \top_1) \qquad \text{for each atomic concept } A \tag{5}$$

$$[U](\mathbf{P} \Rightarrow \top_n) \qquad \text{for each atomic relation } \mathbf{P} \text{ of arity } n \tag{6}$$

$$[U](\sigma(C_1) \Rightarrow \sigma(C_2)) \qquad \text{for each assertion } C_1 \sqsubseteq C_2 \text{ in } S \tag{7}$$

$$[U](\sigma(\mathbf{R}_1) \Rightarrow \sigma(\mathbf{R}_2)) \qquad \text{for each assertion } \mathbf{R}_1 \sqsubseteq \mathbf{R}_2 \text{ in } S \tag{8}$$

The formula (1) above expresses that each state represents an object or a tuple of arity between 2 and n_{max}. The formula (2) expresses that all programs f_i are functional (i.e., deterministic). The formulae (3) and (4) express that the states representing tuples of arity n are exactly those connected through programs $f_1,\ldots,f_n$ to states representing objects, and not connected via programs f_i, with $i > n$, to any state. The formulae (5) and (6) express that states satisfying atomic propositions corresponding to atomic concepts (resp. atomic relations of arity n) are states representing objects (resp. tuples of arity n). Finally, the formulae (7) and (8) encode the assertions in S.

Φ_{conj_j}: encoding of each $conj_j(\vec{\mathbf{a}}, \vec{\mathbf{b}}_j, \vec{\mathbf{c}}_j)$

For each $j \in \{1,\ldots,m\}$, the encoding Φ_{conj_j} of $conj_j(\vec{\mathbf{a}}, \vec{\mathbf{b}}_j, \vec{\mathbf{c}}_j)$ makes use of special atomic propositions, called *name-formulae* whose distinguishing properties are specified by Φ_{aux} (see later). Specifically, one name-formula N_t is introduced for each term t in $\vec{\mathbf{a}}$, $\vec{\mathbf{b}}_j$, $\vec{\mathbf{c}}_j$, and one name-formula $N_{\vec{\mathbf{t}}}$ for each tuple $\vec{\mathbf{t}}$ such that for some $\mathbf{R}$, $\mathbf{R}(\vec{\mathbf{t}})$ appears in $conj_j(\vec{\mathbf{a}}, \vec{\mathbf{b}}_j, \vec{\mathbf{c}}_j)$. A name-formula assigns a name to a term t (resp., tuple $\vec{\mathbf{t}}$), which allows for identifying in a model certain states which

correspond to t (resp., reified counterpart of $\vec{\mathbf{t}}$). The distinguishing properties of name-formulae guarantee that these states share some crucial properties that allow us to isolate a single state as a representative of t (resp., $\vec{\mathbf{t}}$).

Once we have name-formulae in place, we define Φ_{conj_j} as the conjunction of the following formulae:

(1) for each atom $C(t)$ in $conj_j(\vec{\mathbf{a}}, \vec{\mathbf{b}}_j, \vec{\mathbf{c}}_j)$

$$[U](N_t \Rightarrow \sigma(C))$$

(2) for each atom $\mathbf{R}(\vec{\mathbf{t}})$ in $conj_j(\vec{\mathbf{a}}, \vec{\mathbf{b}}_j, \vec{\mathbf{c}}_j)$

$$[U](N_{\vec{\mathbf{t}}} \Rightarrow \sigma(\mathbf{R}))$$

Intuitively, each formula (1) and (2) expresses that the states satisfying the name-formula corresponding to a term (resp., tuple) appearing in an atom, satisfy also the formula corresponding to the predicate of the atom.

$\Phi_{conj'_j}$: encoding of each $\exists \vec{\mathbf{z}}_j . conj'_j(\vec{\mathbf{a}}, \vec{\mathbf{z}}_j, \vec{\mathbf{c}}'_j)$

Now consider a $j \in \{1, \ldots, m'\}$. We construct the formula $\Phi_{conj'_j}$ as a disjunction of formulae, one for each possible partition of the variables $\vec{\mathbf{z}}_j$ in $\exists \vec{\mathbf{z}}_j . conj'_j(\vec{\mathbf{a}}, \vec{\mathbf{z}}_j, \vec{\mathbf{c}}'_j)$. More precisely, to build one such formula, we consider a partition π of the variables $\vec{\mathbf{z}}_j$. Then, for each equivalence class in the partition we choose a variable as a representative, and substitute in $\exists \vec{\mathbf{z}}_j . conj'_j(\vec{\mathbf{a}}, \vec{\mathbf{z}}_j, \vec{\mathbf{c}}'_j)$ all other variables in the same equivalence class by the representative, thus obtaining a formula $\exists \vec{\mathbf{w}}_\pi . conj'_j(\vec{\mathbf{a}}, \vec{\mathbf{w}}_\pi, \vec{\mathbf{c}}'_j)$. Now, from such a formula we build a corresponding CPDL$_g$ formula by making use of a special graph, called *tuple-graph*, which intuitively reflects the dependencies between variables and tuples resulting from the appearance of the variables in the atoms of $\exists \vec{\mathbf{w}}_\pi . conj'_j(\vec{\mathbf{a}}, \vec{\mathbf{w}}_\pi, \vec{\mathbf{c}}'_j)$[5]. A tuple-graph is a directed graph with nodes labeled by CPDL$_g$ formulae and edges labeled by CPDL$_g$ programs, formed as follows:

—There is one node t for each term t in $\vec{\mathbf{a}}$, $\vec{\mathbf{w}}$, $\vec{\mathbf{c}}_j$, and one node $\vec{\mathbf{t}}$ for each tuple $\vec{\mathbf{t}}$ such that $\mathbf{R}(\vec{\mathbf{t}})$ appears in $\exists \vec{\mathbf{w}}_\pi . conj'_j(\vec{\mathbf{a}}, \vec{\mathbf{w}}_\pi, \vec{\mathbf{c}}'_j)$. Each node t is labeled by all $\sigma(C)$ such that $C(t)$ appears in $\exists \vec{\mathbf{w}}_\pi . conj'_j(\vec{\mathbf{a}}, \vec{\mathbf{w}}_\pi, \vec{\mathbf{c}}'_j)$. Each node $\vec{\mathbf{t}}$ is labeled by all $\sigma(\mathbf{R})$ such that $\mathbf{R}(\vec{\mathbf{t}})$ appears in $\exists \vec{\mathbf{w}}_\pi . conj'_j(\vec{\mathbf{a}}, \vec{\mathbf{w}}_\pi, \vec{\mathbf{c}}'_j)$.

—There is one edge labeled by f_i from the node $\vec{\mathbf{t}} = (t_1, \ldots, t_n)$ to the node t_i, $i \in \{1, \ldots, n\}$, for each tuple $\vec{\mathbf{t}}$ such that $\mathbf{R}(\vec{\mathbf{t}})$ appears in $\exists \vec{\mathbf{w}}_\pi . conj'_j(\vec{\mathbf{a}}, \vec{\mathbf{w}}_\pi, \vec{\mathbf{c}}'_j)$.

Notice that dividing the variables $\vec{\mathbf{z}}_j$ in $\exists \vec{\mathbf{z}}_j . conj'_j(\vec{\mathbf{a}}, \vec{\mathbf{z}}_j, \vec{\mathbf{c}}'_j)$ in all possible ways into equivalence classes and replacing equivalent variables by one representative, corresponds to introducing in all possible ways equalities between variables. Such equalities allow us to take into account that a cycle in the tuple graph can in fact be eliminated, and become simply a chain, when different variables are assigned the same object. As will become clear in the following, the distinction between variables appearing in cycles in the tuple-graph and those that do not, is indeed

[5]The tuple-graph is similar to the graph used in [Chekuri and Rajaraman 1997] to detect cyclic dependencies between variables.

necessary for the correctness of the proposed technique for query containment under constraints.

In the following, we call *formula-template* a CPDL$_g$ formula in which formula-placeholders occur that later will be substituted by actual formulae. From the tuple-graph G of $\exists \vec{\mathbf{w}}_\pi . conj'_j(\vec{\mathbf{a}}, \vec{\mathbf{w}}_\pi, \vec{\mathbf{c}}'_j)$, we build a CPDL$_g$ formula-template δ, and to do so we have to consider that in general the tuple-graph is composed of several connected components. For the i-th connected component we build a formula-template δ_i by choosing a starting node t_0 (corresponding to a term) and performing a depth-first visit of the corresponding component and building the formula in a postorder fashion. We describe the construction by defining a visiting function V, which, given a node of the tuple-graph G, returns the corresponding formula-template, and as a side effect marks the nodes of the graph that it visits.

—If $u = t$, then $V(t)$ marks t, and returns the conjunction of:
  (i) t itself, used as a placeholder, and every formula labeling the node t;
  (ii) for each edge $(\vec{\mathbf{t}}, t)$ labeled by f_i (i.e., $t = t_i$ in $\vec{\mathbf{t}}$) such that $\vec{\mathbf{t}}$ is not marked yet, the formula $\langle f_i^- \rangle V(\vec{\mathbf{t}})$.

—If $u = \vec{\mathbf{t}} = (t_1, \ldots, t_n)$, then $V(\vec{\mathbf{t}})$ marks $\vec{\mathbf{t}}$, and returns the conjunction of:
  (i) $\vec{\mathbf{t}}$ itself, used as a placeholder, and every formula labeling the node $\vec{\mathbf{t}}$;
  (ii) for each edge $(\vec{\mathbf{t}}, t_i)$ labeled by f_i, such that t_i is not marked yet, the formula $\langle f_i \rangle V(t_i)$;
  (iii) for each edge $(\vec{\mathbf{t}}, t_i)$ labeled by f_i, such that t_i is already marked, the formula $\langle f_i \rangle t_i$.

Then the formula-template δ_i for the i-th connected component is defined as $V(t_0)$, where t_0 is the starting node chosen for the visit.

The formula-template δ for the whole tuple-graph G, composed of $\ell \geq 1$ connected components, is

$$\langle U \rangle \delta_1 \wedge \cdots \wedge \langle U \rangle \delta_\ell$$

where $\delta_1, \ldots, \delta_\ell$ are the formula-templates corresponding to all the connected components in the tuple-graph G.

We next define the notion of G-substitution for the formula-template δ, which allows us to obtain from δ a CPDL$_g$ formula. Let G be the tuple-graph for $\exists \vec{\mathbf{w}}_\pi . conj'_j(\vec{\mathbf{a}}, \vec{\mathbf{w}}_\pi, \vec{\mathbf{c}}'_j)$, and δ as above, a *G-substitution for δ* is a substitution for the placeholders of δ that replaces:

(i) each placeholder $\vec{\mathbf{t}}$ by $\top_n$, where n is the arity of the tuple $\vec{\mathbf{t}}$;
(ii) each placeholder t in $\vec{\mathbf{a}}$, $\vec{\mathbf{c}}_j$ by the name-formula N_t;
(iii) each placeholder w corresponding to a variable not occurring in a cycle in the tuple-graph by $\top_1$;
(iv) each placeholder w corresponding to a variable occurring in a cycle in the tuple-graph by one of the name-formulae N_t corresponding to a term in $\vec{\mathbf{a}}$, $\vec{\mathbf{b}}_1, \ldots, \vec{\mathbf{b}}_m$, $\vec{\mathbf{c}}_1, \ldots, \vec{\mathbf{c}}_m$ occurring in q.

Notice that there are several G-substitutions for a formula-template δ, one for each possible way of choosing (possibly Skolem) constants according to (iv). Hence, the number of such G-substitutions is $O(\ell_1^{\ell'_1{}^2})$, where ℓ_1 is the number of variables

and constants in q, and ℓ_2' is the number of variables occurring in a cycle in the tuple-graph G.

We are now ready to define the CPDL$_g$ formula φ_π corresponding to a partition π of the variables $\vec{\mathbf{z}}_j$: such a formula φ_π is the disjunction of all formulae obtained by applying to the formula-template δ a possible G-substitution (note that both δ and G are specific for the partition π.)

Since φ_π corresponds to one possible partition of the variables $\vec{\mathbf{z}}_j$, we obtain the formula $\Phi_{conj_j'}$ as the disjunction of all formulae φ_π, one for each possible partition π of the variables $\vec{\mathbf{z}}_j$. The number of such disjuncts is $O(2^{\ell_2})$, where ℓ_2 is the number of variables $\vec{\mathbf{z}}_j$. Therefore, the total number of disjuncts for $\Phi_{conj_j'}$ is $O(\ell_1^{O(\ell_2)})$.

Φ_{aux}: encoding of constants and variables

Let $\Phi' = \Phi_{\mathcal{S}} \wedge (\bigvee_{j=1}^{m} \Phi_{conj_j}) \wedge (\bigwedge_{j=1}^{m'} \neg\Phi_{conj_j'})$. Φ_{aux} is formed by the conjunction of:

(1) the formula $\langle create\rangle N$, for each name-formula N appearing in Φ', which expresses the existence of a state satisfying N;

(2) one formula of the form $[U](N_{c_i} \Rightarrow \neg N_{c_j})$ for each pair of distinct constants c_i, c_j appearing in the queries (not Skolem constants);

(3) for each name-formula $N_{\vec{\mathbf{t}}}$ corresponding to a tuple $\vec{\mathbf{t}} = (t_1, \ldots, t_n)$ appearing in $\bigvee_{j=1}^{m} \Phi_{conj_j}$ (observe that we do not have any name-formula for tuples in $\bigwedge_{j=1}^{m'} \neg\Phi_{conj_j'}$)

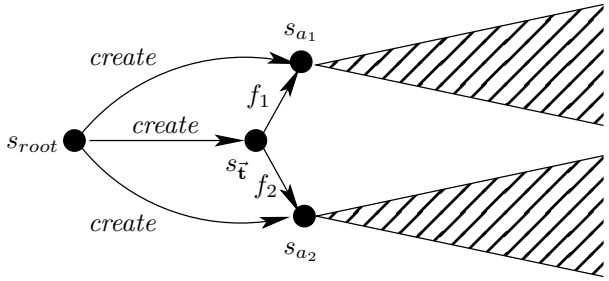
$$[U](N_{\vec{\mathbf{t}}} \equiv \langle f_1\rangle N_{t_1} \wedge \cdots \wedge \langle f_n\rangle N_{t_n} \wedge [f_{n+1}]\mathsf{F})$$
$$[U](N_{t_i} \Rightarrow (\langle f_i^-\rangle N_{\vec{\mathbf{t}}} \wedge [f_i^-]_{\leq 1} N_{\vec{\mathbf{t}}})) \qquad \text{for each } i \in \{1, \ldots, n\}$$

these conjuncts express the relationships between the name-formulae for tuples and those for tuple components;

(4) one formula of the form $[U](N \wedge \phi \Rightarrow [U](N \Rightarrow \phi))$ for each name-formula N and each formula ϕ such that:
—$\phi \in CL(\Phi')$,
—$\phi = \langle \overline{r}\rangle \phi'$ with $\langle r\rangle \phi' \in CL(\Phi')$, and
—$\phi = \langle \overline{r}'; p\rangle N'$ with $r' \in Pre(r)$, $p = f \mid f^-$, and r, f, N' occurring in $CL(\Phi')$,
where $\overline{r}$ is defined inductively as follows:

$$\overline{p} = p; (\wedge_i \neg N_i)?$$
$$\overline{r_1; r_2} = \overline{r_1}; \overline{r_2}$$
$$\overline{r_1 \cup r_2} = \overline{r_1} \cup \overline{r_2}$$
$$\overline{r_1^*} = \overline{r_1}^*$$
$$\overline{\phi?} = \phi?$$

The role of the conjuncts (4) in Φ_{aux} is to enforce that, in every model of $\Phi_{S \not\models q \subseteq q'}$, for each name-formula N, one representative state can be singled out among those satisfying N. This would be trivially obtained if we could force all these states to satisfy exactly the same formulae of the logic. Φ_{aux} forces a weaker condition, namely that these states satisfy the same formulae in the finite set (whose size is



$\Phi_{S \not\models q \subseteq q'}$	is true in	s_{root}
p	is true in	$s_{\vec{\mathbf{t}}}$
$N_{\vec{\mathbf{t}}}$	is true only in	$s_{\vec{\mathbf{t}}}$
N_{a_1}	is true only in	s_{a_1}
N_{a_2}	is true only in	s_{a_2}

Fig. 5. A model of $\Phi_{S \not\models q \subseteq q'}$

polynomial wrt Φ') described above. Theorem 3.3 shows that this is sufficient for our purposes.

Example

We illustrate the encoding of the containment problem $\mathcal{S} \models q \subseteq q'$ into unsatisfiability of the CPDL$_g$ formula $\Phi_{S \not\models q \subseteq q'}$ by means of a simple example.

Consider two queries

$$q(x_1, x_2) \leftarrow p(x_1, x_2)$$
$$q'(x_1, x_2) \leftarrow r(x_1, x_2, z)$$

over a schema $\mathcal{S}$ such that $\mathcal{S} \not\models q \subseteq q'$. Let $\Phi_{\mathcal{S}}$ be the formula encoding $\mathcal{S}$, Φ_{conj} the formula encoding $p(a_1, a_2)$, where a_1, a_2 are two Skolem constants, and Φ_{aux} as specified above. Figure 5 schematically shows a model of the formula $\Phi_{S \not\models q \subseteq q'}$ that represents a counterexample to the containment. The model contains a state $s_{\vec{\mathbf{t}}}$, where $\vec{\mathbf{t}} = (a_1, a_2)$, in which p holds, that, being connected to s_{a_1} and s_{a_2} by means of f_1 and f_2, respectively, represents the tuple $\vec{\mathbf{t}} = (a_1, a_2)$, which satisfies p. The objects s_{a_1} and s_{a_2} satisfy the name-formulae N_{a_1} and N_{a_2}, respectively, and there are no other objects satisfying such name-formulae. From s_{a_1} and s_{a_2} the model has a tree-like structure. We will call such model pseudo-tree admissible (see later). Note that Φ_{conj} is satisfied due to the part of the model involving s_{a_1}, s_{a_2} and $s_{\vec{\mathbf{t}}}$, while Φ_{aux} is trivially satisfied since N_{a_1}, N_{a_2} and $N_{\vec{\mathbf{t}}}$ are true in just one state of the model.

Now, consider the formula-template for $\exists z.r(a_1, a_2, z)$:

$$\langle U \rangle (a_1 \wedge \langle f_1^- \rangle (\vec{\mathbf{t}}' \wedge r \wedge \langle f_2 \rangle a_2 \wedge \langle f_3 \rangle z))$$

where $\vec{\mathbf{t}}' = (a_1, a_2, z)$. From such a formula-template we get the formula $\Phi_{conj'} = \langle U \rangle (N_{a_1} \wedge \langle f_1^- \rangle (\vec{\mathbf{t}}' \wedge r \wedge \langle f_2 \rangle N_{a_2} \wedge \langle f_3 \rangle \top_1))$. Note that, $s_{\vec{\mathbf{t}}}$ has no outgoing f_3 edge, and hence $\top_3$ is false in $s_{\vec{\mathbf{t}}}$; also N_{a_1} and N_{a_2} are true respectively in s_{a_1} and s_{a_2} only. Hence, the formula $\neg\Phi_{conj'} = [U](N_{a_1} \Rightarrow [f_1^-]((\top_3 \wedge r) \Rightarrow ([f_2]N_{a_2} \wedge [f_3]\neg\top_1)))$

is true in s_{root}. Therefore, in the model there is no state that both represents a tuple (a_1, a_2, z), for some z, and that satisfies r.

3.3 Correctness of the Reduction

We now prove the correctness of the reduction of query containment to satisfiability in CPDL$_g$ based on the encoding presented above. We first prove soundness, i.e., if $\Phi_{S \not\models q \subseteq q'}$ is unsatisfiable, then $\mathcal{S} \models q \subseteq q'$ (Theorem 3.2), and then completeness, i.e., if $\Phi_{S \not\models q \subseteq q'}$ is satisfiable, then $\mathcal{S} \not\models q \subseteq q'$ (Theorems 3.3 and 3.4).

To prove soundness, we need a preliminary notion. We say that a tuple-graph G is *satisfied* in an interpretation $\mathcal{M}$ for $\Phi_{S \not\models q \subseteq q'}$ if there exists a *G-$\mathcal{M}$-homomorphism* η, i.e., a mapping from the nodes of G to states of $\mathcal{M}$ such that:

—if a node u of G is a (possibly Skolem) constant, then $\eta(u) \in N_u^{\mathcal{M}}$;
—if a node u of G is labeled by a formula ϕ, then $\eta(u) \in \phi^{\mathcal{M}}$;
—if an edge (u, u') of G is labeled by a program f then $(\eta(u), \eta(u')) \in f^{\mathcal{M}}$.

Given a formula-template δ corresponding to a tuple-graph G and a G-substitution θ of its placeholders, we denote by $\delta\theta$ the formula obtained from δ by substituting the placeholders according to θ.

LEMMA 3.1. *Let G be a tuple-graph, δ the corresponding formula-template, and $\mathcal{M}$ a* CPDL$_g$ *interpretation. If there exists a G-substitution θ such that $(\delta\theta)^{\mathcal{M}}$ is not empty, then G is satisfied in $\mathcal{M}$.*

PROOF. If $(\delta\theta)^{\mathcal{M}}$ is not empty, then it is possible to define a mapping η as follows: for each node t (resp., $\vec{\mathbf{t}}$) in G, let s_t (resp. $s_{\vec{\mathbf{t}}}$) be the state of $\mathcal{M}$ that is used in satisfying $\delta\theta$ in the position corresponding to t (resp. $\vec{\mathbf{t}}$); then $\eta(t) = s_t$ (resp. $\eta(\vec{\mathbf{t}}) = s_{\vec{\mathbf{t}}}$). We show that η is a G-$\mathcal{M}$-homomorphism:

—If t is a (possibly Skolem) constant, then in $\delta\theta$ we have substituted the placeholder t with N_t, and hence $s_t \in N_t^{\mathcal{M}}$.
—If t (resp., $\vec{\mathbf{t}}$) is labeled by ϕ in G, then in δ, in conjunction with the placeholder t there is the formula ϕ, and hence $s_t \in \phi^{\mathcal{M}}$ (resp., $s_{\vec{\mathbf{t}}} \in \phi^{\mathcal{M}}$).
—Finally, if an edge $(\vec{\mathbf{t}}, t)$ is labeled by a program f_i in G, then in δ either in conjunction with t we have $\langle f_i^- \rangle \vec{\mathbf{t}}$, or in conjunction with $\vec{\mathbf{t}}$ we have $\langle f_i \rangle t$, and hence $(s_{\vec{\mathbf{t}}}, t) \in f_i^{\mathcal{M}}$.

□

THEOREM 3.2. *Let $\mathcal{S}$ be a schema, q, q' two queries of the same arity, and $\Phi_{S \not\models q \subseteq q'}$ the formula obtained as specified above. If $\Phi_{S \not\models q \subseteq q'}$ is unsatisfiable, then $\mathcal{S} \models q \subseteq q'$.*

PROOF. We show that, if $\mathcal{S} \not\models q \subseteq q'$ then $\Phi_{S \not\models q \subseteq q'}$ is satisfiable. To this end, we consider a model $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ of $\mathcal{S}$ that makes the following formula true:

$$(conj_1(\vec{\mathbf{a}}, \vec{\mathbf{b}}_1, \vec{\mathbf{c}}_1) \vee \cdots \vee conj_m(\vec{\mathbf{a}}, \vec{\mathbf{b}}_m, \vec{\mathbf{c}}_m)) \wedge$$
$$\neg \exists \vec{\mathbf{z}}_1 . conj'_1(\vec{\mathbf{a}}, \vec{\mathbf{z}}_1, \vec{\mathbf{c}}'_1) \wedge \cdots \wedge \neg \exists \vec{\mathbf{z}}_{m'} . conj'_{m'}(\vec{\mathbf{a}}, \vec{\mathbf{z}}_{m'}, \vec{\mathbf{c}}'_{m'})$$

From $\mathcal{I}$, we build a reified CPDL$_g$ interpretation $\mathcal{M} = (S, \cdot^{\mathcal{M}})$ for $\Phi_{S \not\models q \subseteq q'}$ as follows:

(a) $S = \Delta^{\mathcal{I}} \cup \{s_{root}\} \cup \bigcup_{n\in\{2,\ldots,n_{max}\}} \{s_{\vec{\mathbf{d}}} \mid \vec{\mathbf{d}} \in \top_n^{\mathcal{I}}\}$;
(b) $\top_n^{\mathcal{M}} = \{s_{(d_1,\ldots,d_n)} \mid (d_1,\ldots,d_n) \in \top_n^{\mathcal{I}}\}$, for each $n \in \{2,\ldots,n_{max}\}$;
(c) $f_i^{\mathcal{M}} = \{(s_{(d_1,\ldots,d_n)}, d_i) \mid (d_1,\ldots,d_n) \in \top_n^{\mathcal{I}}\}$, for each $n \in \{2,\ldots,n_{max}\}$ and each $i \in \{1,\ldots,n\}$;
(d) $\mathbf{P}^{\mathcal{M}} = \{s_{(d_1,\ldots,d_n)} \mid (d_1,\ldots,d_n) \in \mathbf{P}^{\mathcal{I}}\}$, for each atomic relation $\mathbf{P}$;
(e) $\top_1^{\mathcal{M}} = \Delta^{\mathcal{I}} \cup \{s_{root}\}$;
(f) $A^{\mathcal{M}} = A^{\mathcal{I}}$, for each atomic concept A;
(g) $N_t^{\mathcal{M}} = \{d\}$, for the element domain $d = t^{\mathcal{I}}$, for each (possibly Skolem) constant t in $\vec{\mathbf{a}}$, $\vec{\mathbf{b}}_1,\ldots,\vec{\mathbf{b}}_m$, $\vec{\mathbf{c}}_1,\ldots,\vec{\mathbf{c}}_m$, $\vec{\mathbf{c}}'_1,\ldots,\vec{\mathbf{c}}'_m$;
(h) $N_{\vec{\mathbf{t}}}^{\mathcal{M}} = \{s_{\vec{\mathbf{d}}}\}$, for the tuple $\vec{\mathbf{d}} = \vec{\mathbf{t}}^{\mathcal{I}}$ of element domains, for each tuple $\vec{\mathbf{t}}$ of (possibly Skolem) constants occurring in $conj_1(\vec{\mathbf{a}}, \vec{\mathbf{b}}_1, \vec{\mathbf{c}}_1) \vee \cdots \vee conj_m(\vec{\mathbf{a}}, \vec{\mathbf{b}}_m, \vec{\mathbf{c}}_m)$;
(i) $create^{\mathcal{M}} = \{(s_{root}, s) \mid s \in N^{\mathcal{M}}$, for some name formula $N\}$.

Next we show that $\Phi_{S\not\models q\subseteq q'}$ is satisfiable by showing that $s_{root} \in \Phi^{\mathcal{M}}_{S\not\models q\subseteq q'}$.

—By construction of $\mathcal{M}$, the interpretation of $\top_n$ (items (b) and (e)), for $n \in \{1,\ldots,n_{max}\}$, and the interpretation of f_i (item (c)), for $i \in \{2,\ldots,n_{max}\}$, satisfy the conjuncts (1)–(4) of $\Phi_{\mathcal{S}}$, while the interpretation of the atoms $\mathbf{P}$ and A (items (d) and (f)) corresponding to atomic relations and atomic concepts, respectively, satisfy the conjuncts (5)–(8), considering that $\mathcal{I}$ is a model of $\mathcal{S}$. Hence, we have that all states of $\mathcal{M}$, including s_{root}, satisfy $\Phi_{\mathcal{S}}$.
—By items (g) and (h), all name-formulae are interpreted as singletons, hence part (4) of the definition of Φ_{aux} is trivially satisfied; by items (g), (h), and (c), part (3) is satisfied; the interpretation of constants in $\mathcal{I}$ guarantees that the corresponding name-formulae are disjoint in $\mathcal{M}$ (item (g)), hence part (2) is also satisfied; finally, by item (i), all instances of the name-formulae are connected via *create* to s_{root}, hence part (1) is satisfied as well.
—By items (f) and (d), conditions (1) and (2) of the definition of Φ_{conj_j} are satisfied if and only if $conj_j(\vec{\mathbf{a}}, \vec{\mathbf{b}}_j, \vec{\mathbf{c}}_j)$ is true in $\mathcal{I}$; hence, considering that $conj_1(\vec{\mathbf{a}}, \vec{\mathbf{b}}_1, \vec{\mathbf{c}}_1) \vee \cdots \vee conj_m(\vec{\mathbf{a}}, \vec{\mathbf{b}}_m, \vec{\mathbf{c}}_m)$ is true $\mathcal{I}$, we have that $s_{root} \in (\bigvee_{j=1}^{m} \Phi_{conj_j})^{\mathcal{M}}$.

It remains to show that $s_{root} \notin \Phi^{\mathcal{M}}_{conj'_j}$, for each $j \in \{1,\ldots,m'\}$. Suppose not, that is, suppose that $s_{root} \in \Phi^{\mathcal{M}}_{conj'_j}$, for some $j \in \{1,\ldots,m'\}$. Then there exists a partition π of the variables $\vec{\mathbf{z}}_j$ in $\exists\vec{\mathbf{z}}_j.conj'_j(\vec{\mathbf{a}}, \vec{\mathbf{z}}_j, \vec{\mathbf{c}}'_j)$ such that $s_{root} \in \varphi^{\mathcal{M}}_{\pi}$, where φ_π is the formula obtained by considering the formula-template δ_π corresponding to the tuple-graph G associated to $\exists\vec{\mathbf{z}}_j.conj'_j(\vec{\mathbf{a}}, \vec{\mathbf{z}}_j, \vec{\mathbf{c}}'_j)$ and applying to it a G-substitution θ, i.e., $\varphi_\pi = \delta_\pi\theta$. Then we have $s_{root} \in (\delta\theta)^{\mathcal{M}}$, and by Lemma 3.1 this implies that the tuple-graph G is satisfied in $\mathcal{M}$. Hence, by items (b)–(g) of the construction of $\mathcal{M}$, we would get that $\exists\vec{\mathbf{z}}_j.conj'_j(\vec{\mathbf{a}}, \vec{\mathbf{z}}_j, \vec{\mathbf{c}}'_j)$ is true in $\mathcal{I}$, contradicting the fact that $\neg\exists\vec{\mathbf{z}}_j.conj'_j(\vec{\mathbf{a}}, \vec{\mathbf{z}}_j, \vec{\mathbf{c}}'_j)$ is true in $\mathcal{I}$. □

Next, we prove completeness of the reduction. In particular, we show that if $\Phi_{S\not\models q\subseteq q'}$ is satisfiable, then it has a model of a specific form (called *pseudo-tree admissible*), from which a model of $\mathcal{S}$ that is a counterexample to the containment can be derived.

We need to introduce the following notions. We say that a model of $\Phi_{S \not\models q \subseteq q'}$ is *tuple-admissible* if there is no pair of states that represent the same reified tuple. We say that a model of $\Phi_{S \not\models q \subseteq q'}$ is *admissible* if it is tuple-admissible and each name-formula is true in exactly one state. We say that a model $\mathcal{M} = (S, \cdot^{\mathcal{M}})$ of $\Phi_{S \not\models q \subseteq q'}$ is a *pseudo-tree admissible model* if it is admissible and has the following form:

—it has a distinguished state s_{root}, and K not necessarily distinct states $s_{N_1}, \ldots, s_{N_K}$, one for each name-formula N_i, such that $N_i^{\mathcal{M}} = \{s_{N_i}\}$;
—$create^{\mathcal{M}} = \{(s_{root}, s_{N_i}) \mid i \in \{1, \ldots, K\}\}$;
—each maximal connected component of $\mathcal{M} \setminus (\{s_{root}\} \cup \{s_{N_i} \mid i \in \{1, \ldots, K\}\})$ is a tree, when viewed as an undirected graph.

Notice that, the subgraph induced by $\mathcal{M} \cap \{s_{N_i} \mid i \in \{1, \ldots, K\}\}$ is an arbitrary graph, instead.

The following theorem shows that, w.r.t. satisfiability of $\Phi_{S \not\models q \subseteq q'}$, one can restrict the attention to pseudo-tree admissible models.

THEOREM 3.3. *Let $\mathcal{S}$ be a schema, q, q' two queries of the same arity, and $\Phi_{S \not\models q \subseteq q'}$ the formula obtained as specified above. If $\Phi_{S \not\models q \subseteq q'}$ is satisfiable then it has a pseudo-tree admissible model.*

PROOF. If $\Phi_{S \not\models q \subseteq q'}$ is satisfiable, then by the tree-model property of CPDL$_g$, $\Phi_{S \not\models q \subseteq q'}$ admits a tree-model $\mathcal{M} = (S, \cdot^{\mathcal{M}})$, in which obviously there is no pair of states that represent the same reified tuple, and is therefore tuple admissible. Let $s_{root} \in \Phi^{\mathcal{M}}$ be the root of $\mathcal{M}$. We transform $\mathcal{M}$ into a new model $\mathcal{M}'' = (S'', \cdot^{\mathcal{M}''})$ with $S'' \subseteq S$, which is pseudo-tree admissible. $\mathcal{M}''$ is obtained from $\mathcal{M}$ by modifying the tree-structure of $\mathcal{M}$ so that all name-formulae are interpreted as singletons.

In particular, to obtain $\mathcal{M}''$ we proceed in two steps. First, we define a model $\mathcal{M}' = (S'', \cdot^{\mathcal{M}''})$ by choosing, for each name-formula N_i, $i \in \{1, \ldots, K\}$, a state s_{N_i}, among the states $s \in N_i^{\mathcal{M}}$ such that $(s_{root}, s) \in create^{\mathcal{M}}$, and then defining:

(a) $create^{\mathcal{M}'} = \{(s_{root}, s_{N_i}) \in create^{\mathcal{M}} \mid i \in \{1, \ldots, K\}\}$
(b) $f^{\mathcal{M}'} = (f^{\mathcal{M}} \setminus (\{(s_{N_i}, s) \in f^{\mathcal{M}} \mid s \in N_j^{\mathcal{M}}, i, j \in \{1, \ldots, K\}\} \cup \{(s, s_{N_j}) \in f^{\mathcal{M}} \mid s \in N_i^{\mathcal{M}}, i, j \in \{1, \ldots, K\}\})) \cup \{(s_{N_i}, s_{N_j}) \mid (s_{N_i}, s) \in f^{\mathcal{M}}, s \in N_j^{\mathcal{M}}, i, j \in \{1, \ldots, K\}\}$,
for each atomic program f corresponding to a tuple-component, i.e., for all atomic programs except *create*,
(c) $A^{\mathcal{M}'} = A^{\mathcal{M}} \cap S'$, for each atomic formula A including name formulae;
(d) $S' = \{s_{root}\} \cup \{s \in S \mid (s_{root}, s) \in create^{\mathcal{M}'} \circ (\bigcup_f (f^{\mathcal{M}'} \cup (f^-)^{\mathcal{M}'}))^*\}$.

The construction above chooses one representative s_{N_i} for each name-formula N_i among those connected through *create* to s_{root} (item (a)). Consider a pair N_1, N_2 of name-formulae related through f. This means that N_1 is a name-formula denoting a tuple $\vec{\mathbf{t}}$, and N_2 is a name-formula denoting a component t of $\vec{\mathbf{t}}$. Consider the representatives s_{N_1} and s_{N_2}. By conjuncts (2) of $\Phi_{\mathcal{S}}$, we have that s_{N_1} has a single f-successor s_2, which is an instance of N_2. Symmetrically, by conjuncts (3) of Φ_{aux}, we have that s_{N_2} has a single f-predecessor s_1 that is an instance of N_1. Item (b)

in the construction above disconnects s_{N_1} from s_2 and s_{N_2} from s_1, thus cutting away the subtrees rooted at s_1 and s_2 (we are only interested in the maximally connected component of the Kripke structure that includes s_{root}, cf. item (d)), and connects s_{N_1} via f to s_{N_2}. Observe that, in doing so, we preserve the number of outgoing f-edges of s_{N_1} and of incoming f-edges of s_{N_2}.

It is possible to show, by using the construction in Lemma 5 of [De Giacomo and Lenzerini 1996][6], that for each $\phi \in CL(\Phi_{S \not\models q \subseteq q'})$ and for each state $s \in \mathcal{S}'$ we have that $s \in \phi^{\mathcal{M}'}$ if and only if $s \in \phi^{\mathcal{M}}$. Hence, since $\Phi_{S \not\models q \subseteq q'} \in CL(\Phi_{S \not\models q \subseteq q'})$, we get that $s_{root} \in (\Phi_{S \not\models q \subseteq q'})^{\mathcal{M}'}$.

From $\mathcal{M}'$ we get $\mathcal{M}''$ by restricting the interpretation of each name-formula N_i to the representative s_{N_i} only, i.e.,

$$N_i^{\mathcal{M}''} = \{s_{N_i}\}$$

Notice that $s_{root} \in \Phi_{\mathcal{S}}^{\mathcal{M}''}$, since name-formulae do not appear in $\Phi_{\mathcal{S}}$. Also, $s_{root} \in (\bigvee_{j=1}^{m} \Phi_{conj_j})^{\mathcal{M}''}$, since $\bigvee_{j=1}^{m} \Phi_{conj_j}$ can be satisfied using only the representatives for name-formulae. Also, $s_{root} \Phi_{aux}^{\mathcal{M}''}$, in particular part (4) is trivially satisfied since in $\mathcal{M}''$ name-formulae are interpreted as singletons. Also, $s_{root} \in (\bigwedge_{j=1}^{m'} \neg\Phi_{conj'_j})^{\mathcal{M}''}$ since this holds already for $\mathcal{M}'$, and each $\neg\Phi_{conj'_j}$ is a disjunction in which name-formulae occur only negatively. Hence, restricting the extension of name-formulae does not restrict the extension of such a formula. □

For pseudo-tree admissible models, one can prove the "converse" of Theorem 3.2.

THEOREM 3.4. *Let $\mathcal{S}$ be a schema, q, q' two queries of the same arity, and $\Phi_{S \not\models q \subseteq q'}$ the formula obtained as specified above. If $\Phi_{S \not\models q \subseteq q'}$ has a pseudo-tree admissible model then $\mathcal{S} \not\models q \subseteq q'$.*

PROOF. We show how to construct from a pseudo-tree admissible model $\mathcal{M}$ of $\Phi_{S \not\models q \subseteq q'}$ a model $\mathcal{I}$ of $\mathcal{S}$ in which there is a tuple $\vec{\mathbf{a}}$ of objects such that $\vec{\mathbf{a}} \in q^{\mathcal{I}}$ and $\vec{\mathbf{a}} \notin q'^{\mathcal{I}}$. $\mathcal{I}$ is built as follows:

—$\Delta^{\mathcal{I}} = \top_1^{\mathcal{M}}$;
—$\mathbf{P}^{\mathcal{I}} = \{(s_1, \ldots, s_n) \mid \exists s' \in \mathbf{P}^{\mathcal{M}}.((s', s_i) \in f_i^{\mathcal{M}}, \text{ for } i \in \{1, \ldots, n\})\}$, for each atomic relation $\mathbf{P}$ of arity n, including $\top_n$;
—$A^{\mathcal{I}} = A^{\mathcal{M}}$, for each atomic concept A, including $\top_1$;
—$t^{\mathcal{I}} = s \in N_t^{\mathcal{M}}$, for each constant and Skolem constant t in q and q'.

To show that $\mathcal{I}$ does the job, we have to show that:

—$\mathcal{I}$ is a model of $\mathcal{S}$;
—$conj_1(\vec{\mathbf{a}}, \vec{\mathbf{b}}_1, \vec{\mathbf{c}}_1) \vee \cdots \vee conj_m(\vec{\mathbf{a}}, \vec{\mathbf{b}}_m, \vec{\mathbf{c}}_m)$ is true in $\mathcal{I}$, i.e., there is one $j \in \{1, \ldots, m\}$ such that $conj_j(\vec{\mathbf{a}}, \vec{\mathbf{b}}_j, \vec{\mathbf{c}}_j)$ is true in $\mathcal{I}$;

[6]The construction in [De Giacomo and Lenzerini 1996] is phrased in the Description Logic $\mathcal{CIQ}$, and it is used to reduce ABox reasoning to satisfiability. $\mathcal{CIQ}$ and CPDL$_g$ can be seen as a syntactic variant one of the other, and our handling of constants, through name-formulae, in CPDL$_g$ is closely related to handling ABoxes in $\mathcal{CIQ}$, the only difference is that for constants in the ABoxes the unique name assumption is made, while here we do not make such an assumption. However, the unique name assumption plays no role in the construction of [De Giacomo and Lenzerini 1996], hence that construction works in our case as well.

—$\neg\exists\vec{\mathbf{z}}_1.conj'_1(\vec{\mathbf{a}},\vec{\mathbf{z}}_1,\vec{\mathbf{c}}'_1) \wedge \cdots \wedge \neg\exists\vec{\mathbf{z}}_{m'}.conj'_{m'}(\vec{\mathbf{a}},\vec{\mathbf{z}}_{m'},\vec{\mathbf{c}}'_{m'})$ is true in $\mathcal{I}$, i.e., for each $j \in \{1,\ldots,m'\}$, we have that $\neg\exists\vec{\mathbf{z}}_j.conj'_j(\vec{\mathbf{a}},\vec{\mathbf{z}}_j,\vec{\mathbf{c}}'_j)$ is true in $\mathcal{I}$.

To show that $\mathcal{I}$ is a model of $\mathcal{S}$ we can exploit the fact that $\mathcal{M} = (S, \cdot^{\mathcal{I}})$ is a model of $\Phi_{\mathcal{S}}$ and that, since it is admissible, there is no pair of states in S that represent the same reified tuple. By construction of $\mathcal{I}$ it is easy to see that all assertions in $\mathcal{S}$ are true in $\mathcal{I}$.

To show that there is one $j \in \{1,\ldots,m\}$ such that $conj_j(\vec{\mathbf{a}},\vec{\mathbf{b}}_j,\vec{\mathbf{c}}_j)$ is true in $\mathcal{I}$, we exploit that $\mathcal{M}$ is an admissible model of $\Phi_{S\not\models q\subseteq q'}$. Hence there is a $j \in \{1,\ldots,m\}$ such that $\mathcal{M}$ is an admissible model of Φ_{conj_j}, and since each name-formula is true in exactly one state, the claim easily follows.

It remains to show that for each $j \in \{1,\ldots,m'\}$, we have that $\neg\exists\vec{\mathbf{z}}_j.conj'_j(\vec{\mathbf{a}},\vec{\mathbf{z}}_j,\vec{\mathbf{c}}'_j)$ is true in $\mathcal{I}$. To do so, we show that, if for some assignment α of the variables $\vec{\mathbf{z}}_j = (z_1,\ldots,z_n)$ to elements $\vec{\mathbf{d}} = (d_1,\ldots,d_n)$ in $\Delta^{\mathcal{I}}$, we have that $conj'_j(\vec{\mathbf{a}},\vec{\mathbf{z}},\vec{\mathbf{c}}_j)$ is true in $\mathcal{I}$, then, we get a contradiction to $\mathcal{M}$ being a model of $\neg\Phi_{conj'_j}$.

By considering which variables have been assigned by α to the same domain elements in $\vec{\mathbf{d}}$, we get a partition π of the variables in $\vec{\mathbf{z}}_j$. Corresponding to such a partition, we have considered in the construction of $\Phi_{conj'_j}$ the formula $\exists\vec{\mathbf{w}}_\pi.conj'_j(\vec{\mathbf{a}},\vec{\mathbf{w}}_\pi,\vec{\mathbf{c}}'_j)$, obtained by replacing all variables in the same equivalence class by a representative. Observe that, as a result, distinct variables in $\vec{\mathbf{w}}_\pi$ are assigned distinct domain elements in $\vec{\mathbf{d}}$.

Observe that, since $\mathcal{M}$ is a pseudo-tree admissible model, the only cycles that can be present in $\mathcal{M}$ are formed by the domain elements that interpret the name-formulae corresponding to the (Skolem) constants in $\vec{\mathbf{a}}$, $\vec{\mathbf{b}}_1,\ldots,\vec{\mathbf{b}}_m$, $\vec{\mathbf{c}}_1,\ldots,\vec{\mathbf{c}}_m$. Let G be the tuple-graph obtained from $\exists\vec{\mathbf{w}}_\pi.conj'_j(\vec{\mathbf{a}},\vec{\mathbf{w}}_\pi,\vec{\mathbf{c}}'_j)$. Since $\mathcal{I}$ keeps the pseudo-tree structure of $\mathcal{M}$, if G contains cycles, the assignment α must assign each variable w in a cycle to an element d that interprets one of the (Skolem) constants t in $\vec{\mathbf{a}}$, $\vec{\mathbf{b}}_1,\ldots,\vec{\mathbf{b}}_m$, $\vec{\mathbf{c}}_1,\ldots,\vec{\mathbf{c}}_m$. It follows, by definition of $\mathcal{I}$, that $d \in N_t^{\mathcal{M}}$.

Now, let δ be the formula-template corresponding to G. From α, we define a G-substitution θ for δ as the G-substitution that replaces (cf. item (iv) in the definition of G-substitution) each placeholder w corresponding to a variable occurring in a cycle in the tuple-graph by the name-formulae N_t, where w is assigned by α to d and $d \in N_t^{\mathcal{M}}$. By considering the construction of the formula-template δ from G and the fact that $\exists\vec{\mathbf{w}}_\pi.conj'_j(\vec{\mathbf{a}},\vec{\mathbf{w}}_\pi,\vec{\mathbf{c}}'_j)$ is true in $\mathcal{I}$ with the assignment α, it can be shown that $s_{root} \in (\delta\theta)^{\mathcal{M}}$. But then, since $\delta\theta$ is one of the disjuncts forming $\Phi_{conj'_j}$, we have that $s_{root} \in (\Phi_{conj'_j})^{\mathcal{M}}$. Hence we get a contradiction. □

The following theorem, which is a consequence of Theorems 3.2, 3.3 and 3.4, summarizes the correctness of our reduction.

THEOREM 3.5. *Let $\mathcal{S}$ be a schema, q, q' two queries of the same arity, and $\Phi_{S\not\models q\subseteq q'}$ the formula obtained as specified above. Then $\mathcal{S} \models q \subseteq q'$ if and only if $\Phi_{S\not\models q\subseteq q'}$ is unsatisfiable.*

3.4 Complexity of Query Containment

Theorem 3.5 directly provides an algorithm for checking query containment in our setting, and therefore shows that the problem is decidable. We analyze now the

computational complexity of such an algorithm for query containment.[7].

THEOREM 3.6. *Let $\mathcal{S}$ be a schema and q and q' two queries. Then, deciding whether $\mathcal{S} \models q \subseteq q'$ can be done in time $2^{p(|\mathcal{S}|+|q|+|q'|\cdot \ell_1^{\ell_2})}$, where $|\mathcal{S}|$, $|q|$, and $|q'|$ are respectively the sizes of $\mathcal{S}$, q, and q', ℓ_1 is the sum of the number of variables and constants in q, and ℓ_2 is the number of existentially quantified variables in q'.*

PROOF. The correctness of the encoding of query containment $\mathcal{S} \models q \subseteq q'$ into unsatisfiability of $\Phi_{S \not\models q \subseteq q'}$ is sanctioned in Theorem 3.5. With regard to complexity, since satisfiability in CPDL$_g$ is EXPTIME-complete, it follows that query containment can be done in time $2^{p(|\Phi_{S \not\models q \subseteq q'}|)}$. It is easy to verify that $|\Phi_{S \not\models q \subseteq q'}| = O(|\mathcal{S}| + |q| + |q'| \cdot \ell_1^{O(\ell_2)})$. □

The previous theorem provides, for the problem of checking whether $\mathcal{S} \models q \subseteq q'$, a single exponential upper bound in the size of $\mathcal{S}$ and of q, and a double exponential upper bound in the size of q' (note that $|q'|$ is an upper bound for ℓ_2). The single exponential upper bound in the size of $\mathcal{S}$ and of q is tight. Indeed, it follows from EXPTIME-hardness of satisfiability in CPDL$_g$ (in fact plain PDL [Fischer and Ladner 1979]) and from the fact that any CPDL$_g$ formula can be expressed as a $\mathcal{DLR}_{reg}$ concept. EXPTIME-hardness in $\mathcal{S}$ holds even in the case where $\mathcal{S}$ does not contain regular expressions. Indeed, the formulae used in the EXPTIME-hardness proof of satisfiability in PDL [Fischer and Ladner 1979], can be expressed as assertions in $\mathcal{DLR}_{reg}$ not involving regular expressions. It is still open whether the double-exponential upper bound in the size of q' is tight.

The double exponential upper bound in the size of q' is due to the exponential blowup in the size of $\Phi_{S \not\models q \subseteq q'}$. By analyzing the reduction presented in Section 3.2, one can observe that such an exponential blowup is only due to those existentially quantified variables in q' that appear inside a cycle in the tuple-graph for q'. Hence, when the tuple-graph for q' does not contain cycles, we have that $|\Phi_{S \not\models q \subseteq q'}| = O(|\mathcal{S}| + |q| + |q'|)$, and query containment can be checked in time $2^{p(|\mathcal{S}|+|q|+|q'|)}$. A relevant case when this occurs is when (the tuple-graph for) the query on the right-hand side has the structure of a tree.

COROLLARY 3.7. *Let $\mathcal{S}$ be a schema, q and q' two queries of the same arity, and let q' have the structure of a tree. Then deciding whether $\mathcal{S} \models q \subseteq q'$ can be done in time $2^{p(|\mathcal{S}|+|q|+|q'|)}$.*

Observe that this gives us an EXPTIME-completeness result for containment of an arbitrary query in a tree-structured one wrt a schema.

Query satisfiability can be considered as a special case of query containment. Indeed, given a schema $\mathcal{S}$, a query q is satisfiable wrt $\mathcal{S}$ if and only if it is not contained in the empty query wrt $\mathcal{S}$. The empty query can be expressed, for example, as $u(\vec{\mathbf{x}}) \leftarrow \mathbf{P}(\vec{\mathbf{x}}) \wedge \neg \mathbf{P}(\vec{\mathbf{x}})$, where $\vec{\mathbf{x}}$ is a tuple of variables and $\mathbf{P}$ is a new atomic relation, both of the same arity as q.

COROLLARY 3.8. *Let $\mathcal{S}$ be a schema, and q a query. Then deciding whether q is satisfiable wrt $\mathcal{S}$ can be done in time $2^{p(|\mathcal{S}|+|q|)}$.*

[7]In the results presented here, we assume unary coding of numbers in number restrictions.

Again, this result shows ExpTime-completeness of query satisfiability wrt a schema.

4. UNDECIDABILITY OF CONTAINMENT OF QUERIES WITH INEQUALITIES

In this section we show that, if we allow for inequalities inside the queries, then query containment wrt a schema becomes undecidable. The proof of undecidability exploits a reduction from the unbounded *tiling problem* [Berger 1966]. An instance $\mathcal{T} = (\mathcal{D}, H, V)$ of the tiling problem is defined by a finite set $\mathcal{D}$ of tile types, a horizontal adjacency relation $H \in \mathcal{D} \times \mathcal{D}$, and a vertical adjacency relation $V \in \mathcal{D} \times \mathcal{D}$, and consists in determining whether there exists a tiling of the first quadrant of the integer plane with tiles of type in $\mathcal{D}$ such that the adjacency conditions are satisfied. As shown e.g., in [Harel 1985; van Emde Boas 1997], the tiling problem is well suited to show undecidability of variants of modal and dynamic logics, and the difficult part of the proof usually consists in enforcing that the tiles lie on an integer grid. To this end we exploit a query containing one inequality.

Formally, given an instance $\mathcal{T} = (\mathcal{D}, H, V)$ of the tiling problem, a *$\mathcal{T}$-tiling* is a total function $t : \mathbb{N} \times \mathbb{N} \longrightarrow \mathcal{D}$, and such a tiling is *correct* if $(t(i,j), t(i+1,j)) \in H$ and $(t(i,j), t(i,j+1)) \in V$, for each $i, j \in \mathbb{N}$. We reduce the problem of checking whether there exists a correct $\mathcal{T}$-tiling to the problem of checking whether $\mathcal{S}_\mathcal{T} \models q_0 \subseteq q_0'$, for suitable schema $\mathcal{S}_\mathcal{T}$ and queries q_0 and q_0' containing inequalities.

Consider an instance $\mathcal{T} = (\mathcal{D}, H, V)$ of the tiling problem with tile types $\mathcal{D} = \{D_1, \ldots, D_k\}$. We construct a schema $\mathcal{S}_\mathcal{T}$ using the atomic concepts *Tile*, $D_1, \ldots, D_k$ and two binary atomic relations *Right* and *Up* as follows:

$$\begin{aligned}
\mathit{Tile} &\sqsubseteq D_1 \sqcup \cdots \sqcup D_k && (9)\\
D_i &\sqsubseteq \mathit{Tile} \qquad \text{for each } i \in \{1, \ldots, k\} && (10)\\
D_i &\sqsubseteq \neg D_j \qquad \text{for each } i, j \in \{1, \ldots, k\},\ i < j && (11)\\
\mathit{Tile} &\sqsubseteq (\leq 1\,[\$1]\,\mathit{Right}) \sqcap (\leq 1\,[\$1]\,\mathit{Up}) && (12)\\
\mathit{Tile} &\sqsubseteq \exists[\$1](\mathit{Right} \sqcap (\$2 : \mathit{Tile})) \sqcap \exists[\$1](\mathit{Up} \sqcap (\$2 : \mathit{Tile})) && (13)\\
D_i &\sqsubseteq \Big(\bigsqcup_{(D_i, D_j) \in H} \neg\exists[\$1](\mathit{Right} \sqcap (\$2 : \neg D_j))\Big) \sqcap && (14)\\
&\quad \Big(\bigsqcup_{(D_i, D_j) \in V} \neg\exists[\$1](\mathit{Up} \sqcap (\$2 : \neg D_j))\Big) \qquad \text{for each } i \in \{1, \ldots, k\}
\end{aligned}$$

Then define the boolean queries q_0 and q_0' as follows:

$$\begin{aligned}
q_0() &\leftarrow \mathit{Tile}(x)\\
q_0'() &\leftarrow \mathit{Right}(x,y) \wedge \mathit{Up}(y,z) \wedge \mathit{Up}(x,y') \wedge \mathit{Right}(y',z') \wedge z \neq z'
\end{aligned}$$

Theorem 4.1. *Let $\mathcal{T}$ be an instance of the tiling problem, $\mathcal{S}_\mathcal{T}$ a schema, and q_0 and q_0' two queries defined as specified above. Then there is a correct $\mathcal{T}$-tiling if and only if $\mathcal{S}_\mathcal{T} \not\models q_0 \subseteq q_0'$.*

Proof. "⇒" Let t be a correct $\mathcal{T}$-tiling. We construct an interpretation $\mathcal{I}_t$ of

$\mathcal{S}_\mathcal{T}$ as follows:

$$\begin{aligned}
\Delta^{\mathcal{I}_t} &= \mathbb{N} \times \mathbb{N} \\
\mathit{Tile}^{\mathcal{I}_t} &= \Delta^{\mathcal{I}_t} \\
\mathcal{D}_h^{\mathcal{I}_t} &= \{(i,j) \in \Delta^{\mathcal{I}_t} \mid t(i,j) = D_h\}, \qquad \text{for each } h \in \{1,\ldots,k\} \\
\mathit{Right}^{\mathcal{I}_t} &= \{((i,j),(i+1,j)) \mid i,j \in \mathbb{N}\} \\
\mathit{Up}^{\mathcal{I}_t} &= \{((i,j),(i,j+1)) \mid i,j \in \mathbb{N}\}
\end{aligned}$$

It is immediate to verify that $\mathcal{I}_t$ is a model of $\mathcal{S}_\mathcal{T}$ and that $q_0^{\mathcal{I}_t}$ is true while ${q'_0}^{\mathcal{I}_t}$ is false.

"$\Leftarrow$" Consider a model $\mathcal{I}$ of $\mathcal{S}_\mathcal{T}$ in which q_0 is true and q'_0 is false. Then $\mathcal{I}$ contains an instance o_0 of *Tile* and assertions (13) in $\mathcal{S}_\mathcal{T}$ force the existence of arbitrary long chains of instances of *Tile*, beginning with o_0 and connected one to the next by alternations of $\mathit{Right}^\mathcal{I}$ and $\mathit{Up}^\mathcal{I}$. By assertions (12), *Right* and *Up* are functional for all instances of *Tile*, and since q'_0 is false in $\mathcal{I}$, these chains of objects form indeed a grid. By assertions (9) and (11), each such object is an instance of precisely one D_h. Hence, we can construct a tiling $t_\mathcal{I}$ by assigning to each object o of the grid, representing an element of the first quadrant, a unique tile type D_h. Considering also assertions (14), it is easy to show by induction on the length of the chain from o_0 to an instance o of *Tile*, that the horizontal and vertical adjacency conditions for o are satisfied. Hence $t_\mathcal{I}$ is a correct $\mathcal{T}$-tiling. □

The theorem above immediately implies undecidability of containment wrt a schema of queries containing inequalities.

THEOREM 4.2. *Let $\mathcal{S}$ be a schema, and q, q' two queries of the same arity that may contain atoms of the form $t \neq t'$. Then the query containment problem $\mathcal{S} \models q \subseteq q'$ is undecidable.*

The reduction used in the proof of Theorem 4.1 shows that query containment remains undecidable even in the restricted case where:

—all relations in $\mathcal{S}$, q, and q' are binary,

—$\mathcal{S}$ does not contain assertions on relations, and all assertions on concepts are of the form $A \sqsubseteq C$,

—$\mathcal{S}$ does not contain regular expressions,

—q and q' do not contain union, or constants expressions, and

—there is a single inequality in q', and no inequality in q.

5. QUERY ANSWERING

As we said in the introduction, it is well known in the database literature that there is a tight connection between the problems of conjunctive query containment and conjunctive query answering [Chandra and Merlin 1977]. Such a relationship has had a particular importance in settings of databases with incomplete information, such as those arising in information integration [Abiteboul and Duschka 1998; Lenzerini 2002], semistructured data [Calvanese et al. 2002], and Description Logics [Baader et al. 2003]. In this section we discuss query answering under Description

Logics constraints, taking advantage of the results on query containment presented above. By query answering under Description Logics constraints we mean to compute the answers to a query over an *incomplete database*, i.e., a database that is partially specified and must satisfy all Description Logic constraints expressed in a schema.[8]

Given a $\mathcal{DLR}_{reg}$ schema $\mathcal{S}$, we specify an incomplete database $\mathcal{D}$ over $\mathcal{S}$ by means of a set of facts, called *membership assertions*, of the form

$$C(a) \qquad\qquad \mathbf{R}(\vec{\mathbf{a}})$$

where C and $\mathbf{R}$ are respectively a concept expression and a relation expression over $\mathcal{S}$, a is a constant, and $\vec{\mathbf{a}}$ is an tuple of constants of the same arity as $\mathbf{R}$. Note that such a notion of incomplete database corresponds to that of ABox in Description Logics [Baader et al. 2003].

An interpretation $\mathcal{I}$ satisfies an assertion $C(a)$ if $a^{\mathcal{I}} \in C^{\mathcal{I}}$, and it satisfies an assertion $\mathbf{R}(\vec{\mathbf{a}})$ if $\vec{\mathbf{a}}^{\mathcal{I}} \in \mathbf{R}^{\mathcal{I}}$. We say that $\mathcal{I}$ is a *model* of $\mathcal{D}$, if it satisfies all assertions in $\mathcal{D}$. An incomplete database $\mathcal{D}$ is *satisfiable with respect to a schema* $\mathcal{S}$ if there is an interpretation $\mathcal{I}$ that is a model of both $\mathcal{S}$ and $\mathcal{D}$. Intuitively, every such interpretation $\mathcal{I}$ represents a complete database that is coherent with both $\mathcal{D}$, and the Description Logic constraints in $\mathcal{S}$.

Given a schema $\mathcal{S}$, an incomplete database $\mathcal{D}$ over $\mathcal{S}$, and a query q for $\mathcal{S}$, the set of *certain answers* $cert(q, \mathcal{S}, \mathcal{D})$ of q with respect to $\mathcal{S}$ and $\mathcal{D}$ is the set of tuples $\vec{\mathbf{c}}$ of constants in $\mathcal{D}$ that are answers to q for all complete databases coherent with $\mathcal{D}$ and $\mathcal{S}$, i.e., such that $\vec{\mathbf{c}} \in q^{\mathcal{I}}$, for all models $\mathcal{I}$ of $\mathcal{S}$ and $\mathcal{D}$.

Given a query

$$q(\vec{\mathbf{x}}) \;\leftarrow\; conj_1(\vec{\mathbf{x}}, \vec{\mathbf{y}}_1, \vec{\mathbf{c}}_1) \vee \cdots \vee conj_m(\vec{\mathbf{x}}, \vec{\mathbf{y}}_m, \vec{\mathbf{c}}_m)$$

in order to check whether a tuple $\vec{\mathbf{c}}$ of constants is in $cert(q, \mathcal{S}, \mathcal{D})$, we can resort to query containment [Abiteboul and Duschka 1998]. In particular, let us define the boolean (i.e., of arity 0) queries $Q_{\mathcal{D}}$ and $Q_{q,\vec{\mathbf{c}}}$ as follows:

$$\begin{array}{l} Q_{\mathcal{D}}() \;\leftarrow\; \bigwedge_{C(a)\in\mathcal{D}} C(a) \wedge \bigwedge_{\mathbf{R}(\vec{\mathbf{a}})\in\mathcal{D}} \mathbf{R}(\vec{\mathbf{a}}) \\ Q_{q,\vec{\mathbf{c}}}() \;\leftarrow\; conj_1(\vec{\mathbf{c}}, \vec{\mathbf{y}}_1, \vec{\mathbf{c}}_1) \vee \cdots \vee conj_m(\vec{\mathbf{c}}, \vec{\mathbf{y}}_m, \vec{\mathbf{c}}_m) \end{array}$$

The first query $Q_{\mathcal{D}}$ is the conjunction of all facts in $\mathcal{D}$, while the second query $Q_{q,\vec{\mathbf{c}}}$ is obtained from q by replacing each variable in $\vec{\mathbf{x}}$ with the corresponding constant in $\vec{\mathbf{c}}$.

THEOREM 5.1. *Let $\mathcal{S}$ be a schema, $\mathcal{D}$ an incomplete database over $\mathcal{S}$, q a query for $\mathcal{S}$, and $\vec{\mathbf{c}}$ a tuple of constants in $\mathcal{D}$ of the same arity as q. Then $\vec{\mathbf{c}} \in cert(q, \mathcal{S}, \mathcal{D})$ if and only if $\mathcal{S} \models Q_{\mathcal{D}} \subseteq Q_{q,\vec{\mathbf{c}}}$.*

PROOF. The result can be proved exactly as in [Abiteboul and Duschka 1998]. □

From Theorem 3.6 we immediately obtain the following complexity result.

THEOREM 5.2. *Let $\mathcal{S}$ be a schema, $\mathcal{D}$ an incomplete database over $\mathcal{S}$, q a query for $\mathcal{S}$, and $\vec{\mathbf{c}}$ a tuple of constants in $\mathcal{D}$ of the same arity as q. Then deciding whether*

[8]Note that, the case in which we have complete information on the database, the constraints do not play any role on query answering, assuming that the database is consistent with them.

$\vec{\mathbf{c}} \in cert(q, \mathcal{S}, \mathcal{D})$ can be done in time $2^{p(|\mathcal{S}|+|\mathcal{D}|+|q|\cdot d^{\ell})}$, where $|\mathcal{S}|$, $|\mathcal{D}|$, and $|q|$ are respectively the sizes of $\mathcal{S}$, $\mathcal{D}$, and q, d is the number of constants in $\mathcal{D}$ and q, and ℓ is the number of existentially quantified variables in q.

Note that this means that, while query answering is double exponential in combined complexity, it is actually only single exponential in the number of constants in the database. It follows, that our technique is exponential in data complexity, i.e., the complexity measured only with respect to the size of $\mathcal{D}$. Note that this is the first result on data complexity on answering unions of conjunctive queries under such expressive Description Logics constraints.

Finally, it follows directly from the semantics, that satisfiability of a given incomplete database $\mathcal{D}$ with respect to a schema $\mathcal{S}$. can be rephrased as satisfiability of the query $Q_{\mathcal{D}}$ with respect to $\mathcal{S}$. Thus, we obtain the following result.

COROLLARY 5.3. *Let $\mathcal{S}$ be a schema and $\mathcal{D}$ an incomplete database over $\mathcal{S}$. Then deciding whether $\mathcal{D}$ is satisfiable with respect to $\mathcal{S}$ can be done in time $2^{p(|\mathcal{S}|+|\mathcal{D}|)}$.*

In Description Logics jargon, this shows EXPTIME-completeness of TBox+ABox satisfiability in our setting. Observe that, since we allow for union of conjunctive queries on the left-hand side query in the containment, this result can be immediately extended to satisfiability of a TBox together with a *disjunction* of ABoxes [Calvanese et al. 2001].

6. CONCLUSIONS

In this paper we have introduced $\mathcal{DLR}_{reg}$, an expressive language for specifying database schemas and non-recursive Datalog queries, and we have presented decidability (with complexity) and undecidability results of both the problem of checking query containment, and the problem of answering queries under the constraints expressed in the schema.

The query language considered in this paper allows no form of recursion, not even the transitive closure of binary relations. It is our aim in the future to extend our analysis to the case where queries may contain regular expressions, in the spirit of [Calvanese et al. 2000].

ACKNOWLEDGMENTS

We thank the anonymous reviewers for many valuable comments, which helped improving the paper. This work has been partially supported by the EU funded FP6-7603 FET Project Thinking ONtologiES (TONES), by project HYPER, funded by IBM through a Shared University Research (SUR) Award grant, and by MIUR FIRB 2005 project "Tecnologie Orientate alla Conoscenza per Aggregazioni di Imprese in Internet" (TOCAI.IT).

REFERENCES

ABITEBOUL, S. AND DUSCHKA, O. 1998. Complexity of answering queries using materialized views. In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'98)*. 254–265.

ABITEBOUL, S., HULL, R., AND VIANU, V. 1995. *Foundations of Databases*. Addison Wesley Publ. Co.

Aho, A. V., Sagiv, Y., and Ullman, J. D. 1979a. Efficient optimization of a class of relational expressions. *ACM Trans. on Database Systems 4*, 297–314.

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

Amir, K., Park, S., Tewari, R., and Padmanabhan, S. 2003. Scalable template-based query containment checking for web semantic caches. In *Proc. of the 19th IEEE Int. Conf. on Data Engineering (ICDE 2003)*. 493–504.

Baader, F., Calvanese, D., McGuinness, D., Nardi, D., and Patel-Schneider, P. F., Eds. 2003. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press.

Berger, R. 1966. The undecidability of the dominoe problem. *Mem. Amer. Math. Soc. 66*, 1–72.

Blackburn, P., de Rijke, M., and Venema, Y. 2001. *Modal Logic*. Cambridge Tracts in Theoretical Computer Science, vol. 53. Cambridge University Press.

Bonatti, P. A. 2004. On the decidability of containment of recursive datalog queries - preliminary report. In *Proc. of the 23rd ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2004)*. 297–306.

Borgida, A. and Brachman, R. J. 2003. Conceptual modeling with description logics. See Baader et al. [2003], Chapter 10, 349–372.

Calvanese, D. and De Giacomo, G. 2003. Expressive description logics. See Baader et al. [2003], Chapter 5, 178–218.

Calvanese, D., De Giacomo, G., and Lenzerini, M. 1995. Structured objects: Modeling and reasoning. In *Proc. of the 4th Int. Conf. on Deductive and Object-Oriented Databases (DOOD'95)*. Lecture Notes in Computer Science, vol. 1013. Springer, 229–246.

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

Calvanese, D., De Giacomo, G., and Lenzerini, M. 2001. Identification constraints and functional dependencies in description logics. In *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001)*. 155–160.

Calvanese, D., De Giacomo, G., Lenzerini, M., and Nardi, D. 2001. Reasoning in expressive description logics. In *Handbook of Automated Reasoning*, A. Robinson and A. Voronkov, Eds. Vol. II. Elsevier Science Publishers, Chapter 23, 1581–1634.

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

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

Calvanese, D., De Giacomo, G., Lenzerini, M., and Vardi, M. Y. 2002. View-based query answering and query containment over semistructured data. In *Revised Papers of the 8th International Workshop on Database Programming Languages (DBPL 2001)*, G. Ghelli and G. Grahne, Eds. Lecture Notes in Computer Science, vol. 2397. Springer, 40–61.

Calvanese, D., De Giacomo, G., and Vardi, M. Y. 2003. Decidable containment of recursive queries. In *Proc. of the 9th Int. Conf. on Database Theory (ICDT 2003)*. Lecture Notes in Computer Science, vol. 2572. Springer, 330–345.

Calvanese, D., Lenzerini, M., and Nardi, D. 1999. Unifying class-based representation formalisms. *J. of Artificial Intelligence Research 11*, 199–240.

Calvanese, D. and Rosati, R. 2003. Answering recursive queries under keys and foreign keys is undecidable. In *Proc. of the 10th Int. Workshop on Knowledge Representation meets Databases (KRDB 2003)*. CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/Vol-79/`.

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

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

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

Chandra, A. K. and Vardi, M. Y. 1985. The implication problem for functional and inclusion dependencies is undecidable. *SIAM J. on Computing 14,* 3, 671–677.

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

Chekuri, C. and Rajaraman, A. 1997. Conjunctive query containment revisited. In *Proc. of the 6th Int. Conf. on Database Theory (ICDT'97)*. 56–70.

De Giacomo, G. and Lenzerini, M. 1994. Boosting the correspondence between description logics and propositional dynamic logics. In *Proc. of the 12th Nat. Conf. on Artificial Intelligence (AAAI'94)*. 205–212.

De Giacomo, G. and Lenzerini, M. 1995. 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)*. 801–807.

De Giacomo, G. and Lenzerini, M. 1996. TBox and ABox reasoning in expressive description logics. In *Proc. of the 5th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'96)*. 316–327.

Dong, G. and Su, J. 1996. Conjunctive query containment with respect to views and constraints. *Information Processing Lett. 57,* 2, 95–102.

Donini, F. M., Lenzerini, M., Nardi, D., and Schaerf, A. 1998. $\mathcal{AL}$-log: Integrating Datalog and description logics. *J. of Intelligent Information Systems 10,* 3, 227–252.

Enderton, H. B. 1972. *A Mathematical Introduction to Logic*. Academic Press.

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

Fischer, M. J. and Ladner, R. E. 1979. Propositional dynamic logic of regular programs. *J. of Computer and System Sciences 18*, 194–211.

Gruber, T. R. 1993. Towards principles for the design of ontologies used for knowledge sharing. In *Formal Ontology in Conceptual Analysis and Knowledge Representation*, N. Guarino and R. Poli, Eds. Kluwer Academic Publishers.

Gupta, A. and Mumick, I. S. 1995. Maintenance of materialized views: Problems, techniques, and applications. *Bull. of the IEEE Computer Society Technical Committee on Data Engineering 18,* 2, 3–18.

Gupta, A., Sagiv, Y., Ullman, J. D., and Widom, J. 1994. Constraint checking with partial information. In *Proc. of the 13th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'94)*.

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

Harel, D. 1985. Recurring dominoes: Making the highly undecidable highly understandable. *Ann. of Discrete Mathematics 24*, 51–72.

Horrocks, I., Sattler, U., Tessaris, S., and Tobies, S. 2000. How to decide query containment under constraints using a description logic. In *Proc. of the 7th Int. Conf. on Logic for Programming and Automated Reasoning (LPAR 2000)*. Lecture Notes in Computer Science, vol. 1955. Springer, 326–343.

Hull, R. 1997. Managing semantic heterogeneity in databases: A theoretical perspective. In *Proc. of the 16th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'97)*. 51–61.

Ioannidis, Y. E. and Ramakrishnan, R. 1995. Containment of conjunctive queries: Beyond relations as sets. *ACM Trans. on Database Systems 20,* 3, 288–324.

Johnson, D. S. and Klug, A. C. 1984. Testing containment of conjunctive queries under functional and inclusion dependencies. *J. of Computer and System Sciences 28,* 1, 167–189.

Klug, A. C. 1988. On conjunctive queries containing inequalities. *J. of the ACM 35,* 1, 146–160.

Kozen, D. and Tiuryn, J. 1990. Logics of programs. In *Handbook of Theoretical Computer Science — Formal Models and Semantics*, J. van Leeuwen, Ed. Elsevier Science Publishers, 789–840.

Lenzerini, M. 2002. Data integration: A theoretical perspective. In *Proc. of the 21st ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2002)*. 233–246.

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

Levy, A. Y., Srivastava, D., and Kirk, T. 1995. Data model and query evaluation in global information systems. *J. of Intelligent Information Systems 5*, 121–143.

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

Mitchell, J. C. 1983. The implication problem for functional and inclusion dependencies. *Information and Control 56*, 154–173.

Neven, F. and Schwentick, T. 2003. XPath containment in the presence of disjunction, DTDs, and variables. In *Proc. of the 9th Int. Conf. on Database Theory (ICDT 2003)*. 315–329.

Patel-Schneider, P., Hayes, P., and Horrocks, I. 2004. OWL Web Ontology Language semantics and abstract syntax. W3C Recommendation. Available at `http://www.w3.org/TR/owl-semantics/`.

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

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

Ullman, J. D. 1997. Information integration using logical views. In *Proc. of the 6th Int. Conf. on Database Theory (ICDT'97)*. Lecture Notes in Computer Science, vol. 1186. Springer, 19–40.

van der Meyden, R. 1998. Logical approaches to incomplete information. In *Logics for Databases and Information Systems*, J. Chomicki and G. Saake, Eds. Kluwer Academic Publishers, 307–356.

van Emde Boas, P. 1997. The convenience of tilings. In *Complexity, Logic, and Recursion Theory*, A. Sorbi, Ed. Lecture Notes in Pure and Applied Mathematics, vol. 187. Marcel Dekker Inc., 331–363.

Widom (ed.), J. 1995. Special issue on materialized views and data warehousing. *Bull. of the IEEE Computer Society Technical Committee on Data Engineering 18,* 2.

Wood, P. T. 2003. Containment for XPath fragments under DTD constraints. In *Proc. of the 9th Int. Conf. on Database Theory (ICDT 2003)*. 300–314.

Received July 2005; revised April 2006; accepted December 2006