On the Decidability of Query Containment under Constraints

Diego Calvanese
Dip. di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, 00198 Roma, Italy
calvanese@dis.uniroma1.it

Giuseppe De Giacomo
Dip. di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, 00198 Roma, Italy
degiacomo@dis.uniroma1.it

Maurizio Lenzerini
Dip. di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, 00198 Roma, Italy
lenzerini@dis.uniroma1.it

Abstract

Query containment under constraints is the problem of checking whether for every database satisfying a given set of constraints, the result of one query is a subset of the result of another query. Recent research points out that this is a central problem in several database applications, and we address it within a setting where constraints are specified in the form of special inclusion dependencies over complex expressions, built by using intersection and difference of relations, special forms of quantification, regular expressions over binary relations, and cardinality constraints. These types of constraints capture a great variety of data models, including the relational, the entity-relational, and the object-oriented model.

We study the problem of checking whether q is contained in q' with respect to the constraints specified in a schema $\mathcal{S}$, where q and q' are nonrecursive Datalog programs whose atoms are complex expressions. We present the following results on query containment. For the case where q does not contain regular expressions, we provide a method for deciding query containment, and analyze its computational complexity. We do the same for the case where neither $\mathcal{S}$ nor q, q' contain number restrictions. To the best of our knowledge, this yields the first decidability result on containment of conjunctive queries with regular expressions. Finally, we prove that the problem is undecidable for the case where we admit inequalities in q'.

1 Introduction

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 recent papers point out that this problem is important in several contexts, including information integration [33], query optimization [2, 3], (materialized) view maintenance [21], data warehousing [36], and constraint checking [22].

We deal with the problem of query containment under constraints, i.e. the one of checking whether containment between two queries holds for every database satisfying a given set of constraints. This problem is relevant in every situation where the database schema is specified with a rich data definition language. In particular in the case of information integration, queries are 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 [7, 24, 33, 8, 29].

The complexity of query containment in the absence of constraints has been studied in various settings. In [10], NP-completeness has been established for conjunctive queries, and in [13] 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 [26, 34], Π_2^p-completeness of containment of conjunctive queries with inequalities was proved, and in [32] 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 [12] and [34], respectively.

Query containment under constraints has also been the subject of several investigations. For example, decidability of conjunctive query containment was investigated in [4] under functional and multi-valued dependencies, in [14] under functional and inclusion dependencies, in [9, 28, 30] under constraints representing *is-a* hierarchies and complex objects, and in [17] in the case of constraints represented as Datalog programs.

In this paper we address query containment 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 logic-based formalism. 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 dependen-

[1] We refer to the set semantics of query containment. Bag semantics is studied, for example, in [25].

cies. It can be shown that our formalism is able to capture a great variety of data models, including the relational, the entity-relational, and the object-oriented model, all extended with various forms of constraints.

- Queries are formed as disjunctions of conjunctive queries whose atoms are complex expressions, and therefore, can express non-recursive Datalog programs. Since complex expressions are the basic building blocks of queries, it is possible to specify queries whose atoms are regular expressions. Recent papers point out that this feature is important in modern query languages (see e.g. [1]).

We observe that, given the form of constraints and queries allowed in our approach, none of the previous results can be applied to get decidability/undecidability of query containment. We adopt a novel technique for addressing the problem, based on translating the schema and the queries into a particular Propositional Dynamic Logic (PDL) formula, and then checking the unsatisfiability of the formula. The technique is justified by the fact that reasoning about the schema itself (without the queries) is optimally done within the framework of PDL [16].

We present the following results on checking whether a query q is contained in another query q' with respect to the constraints specified in a schema $\mathcal{S}$:

1. For the case where q does not contain regular expressions, we provide a method for query containment, thus showing that the problem is decidable, and analyze its computational complexity.

2. We do the same for the case where neither $\mathcal{S}$ nor q, q' contain number restrictions. To the best of our knowledge, this is the first decidability result on containment of conjunctive queries with regular expressions.

3. For the case where we allow inequalities in q', we prove that the problem is undecidable, even for very simple schemas and queries.

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 describe the logic CPDL_g, which will be used for deciding query containment. In Sections 4, 5 and 6 we present the two decidability results, and the undecidability result, respectively. Finally, Section 7 concludes the paper.

2 Schemas and Queries in $\mathcal{DLR}_{reg}$

We use the logical language $\mathcal{DLR}_{reg}$, inspired by [8, 6], to specify database schemas and queries. 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.

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.[2].

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

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

where i and j denote components of relations, i.e. integers between 1 and n_{max}, n denotes the arity of a relation, i.e. an integer between 2 and n_{max}, and k denotes a nonnegative integer.

Expressions of the form $(\leq k\,[\$i]\mathbf{R})$ are called *number restrictions*. 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. Also, we consider only concepts and relations that are *well-typed*, which means that

- only relations of the same arity n are combined to form expressions of type $\mathbf{R}_1 \sqcap \mathbf{R}_2$ (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

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

where $\mathbf{R}_1$ and $\mathbf{R}_2$ are of the same arity.

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 of constants $\mathcal{C}$ (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 of $\Delta^{\mathcal{I}}$ under the unique name assumption

- to each concept C a subset $C^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$

- to each regular expression E a subset $E^{\mathcal{I}}$ of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$

- to each relation $\mathbf{R}$ of arity n a subset $\mathbf{R}^{\mathcal{I}}$ of $(\Delta^{\mathcal{I}})^n$

such 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.

An interpretation $\mathcal{I}$ *satisfies* an assertion $\mathbf{R}_1 \sqsubseteq \mathbf{R}_2$ (resp. $C_1 \sqsubseteq C_2$) if $\mathbf{R}_1^{\mathcal{I}} \subseteq \mathbf{R}_2^{\mathcal{I}}$ (resp. $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$). 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 $\mathcal{S}$.

[2] 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.

$$
\begin{array}{rcl}
\top_n^{\mathcal{I}} & \subseteq & (\Delta^{\mathcal{I}})^n \\
\mathbf{P}^{\mathcal{I}} & \subseteq & \top_n^{\mathcal{I}} \\
(\$i/n : C)^{\mathcal{I}} & = & \{(d_1,\ldots,d_n) \in \top_n^{\mathcal{I}} \mid d_i \in C^{\mathcal{I}}\} \\
(\neg\mathbf{R})^{\mathcal{I}} & = & \top_n^{\mathcal{I}} \setminus \mathbf{R}^{\mathcal{I}} \\
(\mathbf{R}_1 \sqcap \mathbf{R}_2)^{\mathcal{I}} & = & \mathbf{R}_1^{\mathcal{I}} \cap \mathbf{R}_2^{\mathcal{I}}
\end{array}
\qquad
\begin{array}{rcl}
\varepsilon^{\mathcal{I}} & = & \{(x,x) \mid x \in \Delta^{\mathcal{I}}\} \\
(\mathbf{R}|_{\$i,\$j})^{\mathcal{I}} & = & \{(x_i,x_j) \mid (x_1,\ldots,x_n) \in \mathbf{R}^{\mathcal{I}}\} \\
(E_1 \circ E_2)^{\mathcal{I}} & = & E_1^{\mathcal{I}} \circ E_2^{\mathcal{I}} \\
(E_1 \sqcup E_2)^{\mathcal{I}} & = & E_1^{\mathcal{I}} \cup E_2^{\mathcal{I}} \\
(E^*)^{\mathcal{I}} & = & (E^{\mathcal{I}})^*
\end{array}
$$

$$
\begin{array}{rcl}
\top_1^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \qquad\qquad (\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
A^{\mathcal{I}} & \subseteq & \Delta^{\mathcal{I}} \qquad\qquad (C_1 \sqcap C_2)^{\mathcal{I}} = C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}} \\
(\exists E.C)^{\mathcal{I}} & = & \{d \in \Delta^{\mathcal{I}} \mid \exists d' \in C^{\mathcal{I}}.(d,d') \in E^{\mathcal{I}}\} \\
(\exists[\$i]\mathbf{R})^{\mathcal{I}} & = & \{d \in \Delta^{\mathcal{I}} \mid \exists(d_1,\ldots,d_n) \in \mathbf{R}^{\mathcal{I}}.d_i = d\} \\
(\leq k\,[\$i]\mathbf{R})^{\mathcal{I}} & = & \{d \in \Delta^{\mathcal{I}} \mid \#\{(d_1,\ldots,d_n) \in \mathbf{R}_1^{\mathcal{I}} \mid d_i = d\} \leq k\}
\end{array}
$$

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

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 [6], 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, whereas 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 [7], 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, the method described in [6] allows one to verify in EXPTIME whether a given assertion is satisfied in every model of a schema.

It follows from EXPTIME-hardness of satisfiability in PDL and from the fact that any PDL formula can be expressed as a $\mathcal{DLR}_{reg}$ concept, that logical implication in $\mathcal{DLR}_{reg}$ is EXPTIME-hard. This holds even in the case where the schema does not contain regular expressions. Indeed, the formulas used in the EXPTIME-hardness proof of satisfiability in PDL [20], can be expressed as assertions in $\mathcal{DLR}_{reg}$ not involving regular expressions.

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 independently in [31, 11], does not apply.

2.2 Queries

A query q for a $\mathcal{DLR}_{reg}$ schema is a non-recursive Datalog query, written in the form:

$$q(\vec{\mathbf{x}}) \;\leftarrow\; body_1(\vec{\mathbf{x}},\vec{\mathbf{y}}_1,\vec{\mathbf{c}}_1) \vee \cdots \vee body_m(\vec{\mathbf{x}},\vec{\mathbf{y}}_m,\vec{\mathbf{c}}_m)$$

where each $body_i(\vec{\mathbf{x}},\vec{\mathbf{y}}_i,\vec{\mathbf{c}}_i)$ is a conjunction of *atoms*, and $\vec{\mathbf{x}},\vec{\mathbf{y}}_i$ (resp. $\vec{\mathbf{c}}_i$) are all the variables (resp. constants) appearing in the conjunct. Each atom has one of the forms $\mathbf{R}(\vec{\mathbf{t}})$, $C(t)$, or $E(t,t')$, where

- $\vec{\mathbf{t}}$, t, and t' are constants or variables in $\vec{\mathbf{x}},\vec{\mathbf{y}}_i,\vec{\mathbf{c}}_i$
- $\mathbf{R}$, C, and E are relations, concepts, and regular 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 $\mathcal{DLR}_{reg}$ relations and concepts, freely used in the assertions of the schema. This distinguishes our approach with respect to [18, 28], where no constraints can be expressed in the schema on the relations that appear in the queries.

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, when substituting each o_i for x_i, the formula

$$\exists\vec{\mathbf{y}}_1.body_1(\vec{\mathbf{x}},\vec{\mathbf{y}}_1,\vec{\mathbf{c}}_1) \vee \cdots \vee \exists\vec{\mathbf{y}}_m.body_m(\vec{\mathbf{x}},\vec{\mathbf{y}}_m,\vec{\mathbf{c}}_m)$$

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

If q and q' are two queries (of the same arity) for $\mathcal{S}$, we say that q is *contained in* q' wrt $\mathcal{S}$, denoted $\mathcal{S} \models q \subseteq q'$, if $q^{\mathcal{I}} \subseteq q'^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{S}$. Given a $\mathcal{DLR}_{reg}$ schema $\mathcal{S}$ and two queries for $\mathcal{S}$

$$\begin{array}{rcl} q(\vec{\mathbf{x}}) & \leftarrow & body_1(\vec{\mathbf{x}}, \vec{\mathbf{y}}_1, \vec{\mathbf{c}}_1) \vee \cdots \vee body_m(\vec{\mathbf{x}}, \vec{\mathbf{y}}_m, \vec{\mathbf{c}}_m) \\ q'(\vec{\mathbf{x}}) & \leftarrow & body'_1(\vec{\mathbf{x}}, \vec{\mathbf{y}}'_1, \vec{\mathbf{c}}'_1) \vee \cdots \vee body'_{m'}(\vec{\mathbf{x}}, \vec{\mathbf{y}}'_{m'}, \vec{\mathbf{c}}'_{m'}) \end{array}$$

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

$$\begin{array}{l} (body_1(\vec{\mathbf{x}}, \vec{\mathbf{y}}_1, \vec{\mathbf{c}}_1) \vee \cdots \vee body_m(\vec{\mathbf{x}}, \vec{\mathbf{y}}_m, \vec{\mathbf{c}}_m)) \wedge \\ \neg\exists \vec{\mathbf{z}}_1 . body'_1(\vec{\mathbf{x}}, \vec{\mathbf{z}}_1, \vec{\mathbf{c}}'_1) \wedge \cdots \wedge \neg\exists \vec{\mathbf{z}}_{m'} . body'_{m'}(\vec{\mathbf{x}}, \vec{\mathbf{z}}_{m'}, \vec{\mathbf{c}}'_{m'}) \end{array}$$

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

$$\begin{array}{l} (body_1(\vec{\mathbf{a}}, \vec{\mathbf{b}}_1, \vec{\mathbf{c}}_1) \vee \cdots \vee body_m(\vec{\mathbf{a}}, \vec{\mathbf{b}}_m, \vec{\mathbf{c}}_m)) \wedge \\ \neg\exists \vec{\mathbf{z}}_1 . body'_1(\vec{\mathbf{a}}, \vec{\mathbf{z}}_1, \vec{\mathbf{c}}_1) \wedge \cdots \wedge \neg\exists \vec{\mathbf{z}}_{m'} . body'_{m'}(\vec{\mathbf{a}}, \vec{\mathbf{z}}_{m'}, \vec{\mathbf{c}}_{m'}) \end{array}$$

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

The *query containment problem* is the one of checking whether $\mathcal{S} \models q \subseteq q'$, where $\mathcal{S}$, q, q' are given as input. A special case of query containment is *query satisfiability*, which amounts to verify whether a given query is interpreted as the empty set in every model of the schema (note that the query $u(\vec{\mathbf{x}} \leftarrow \mathbf{P}(\vec{\mathbf{x}} \wedge \neg\mathbf{P}(\vec{\mathbf{x}})$ is unsatisfiable).

We observe that, since logical implication in $\mathcal{DLR}_{reg}$ is already EXPTIME-complete, query containment in our setting is EXPTIME-hard, even in the case where neither the schema nor the queries contain regular expressions.

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 all the department that, directly or indirectly, controls it are 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 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:

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

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 the pairs of departments that are controlled by the same department and that comprise at least one sold department. The second query, called q', retrieves all the pair (x, y) of departments such that x has been sold, and y belong to the same `CONTROLS`-tree of x. The queries q and q' are defined as follows:

$$\begin{array}{rcl} q(x, y) & \leftarrow & \texttt{CONTROLS}(z_1, x) \wedge \texttt{CONTROLS}(z_1, y) \wedge \\ & & \texttt{SOLD}(y, z_2, z_3) \\ q'(x, y) & \leftarrow & \texttt{Dept}(x) \wedge \texttt{SOLD}(x, z_1, z_2) \wedge \\ & & (\texttt{CONTROLS}|_{\$1,\$2} \sqcup \texttt{CONTROLS}|_{\$2,\$1})^*(x, y) \end{array}$$

One can verify that $\mathcal{S} \models q \subseteq q'$. This follows from the fact that the regular expression $(\texttt{CONTROLS}|_{\$2,\$1} \circ \texttt{CONTROLS}|_{\$1,\$2})$ is "contained" in the regular expression $(\texttt{CONTROLS}|_{\$1,\$2} \sqcup \texttt{CONTROLS}|_{\$2,\$1})^*$, and from the notion of `CONTROLS`-tree as defined in $\mathcal{S}$.

Also, if we add to $q'(x, y)$ the condition that department y is not sold, we obtain the query

$$\begin{array}{l} q''(x, y) \leftarrow \texttt{Dept}(x) \wedge \texttt{SOLD}(x, z_1, z_2) \wedge \neg\texttt{SOLD}(y, w_1, w_2) \wedge \\ \qquad\qquad (\texttt{CONTROLS}|_{\$1,\$2} \sqcup \texttt{CONTROLS}|_{\$2,\$1})^*(x, y) \end{array}$$

which is unsatisfiable.

3 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 [20]. Since then PDLs have been studied extensively and extended in several ways (see e.g. [27] for a survey).

Here, we make use of CPDL_g (studied in [16] in the context of description logics), which is an extension of Converse PDL [27] with *graded modalities* [19]. 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*):

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

The abbreviation $[r]\phi$ is used 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 $\cdot^{\mathcal{M}}$ formulae as subsets of S and programs as binary relations over S. The semantics of each construct is reported below:

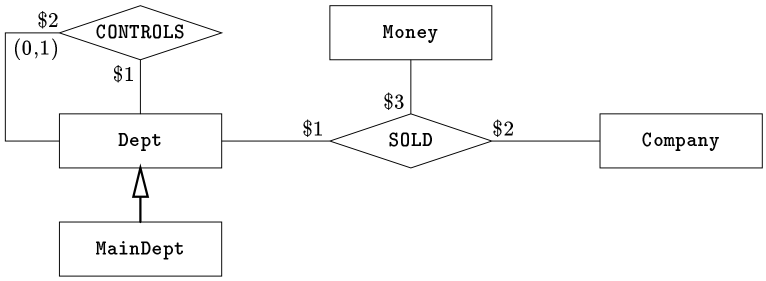


Figure 2: The entity-relationship diagram for the example in Section 2.3

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

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

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 decidability* of checking satisfiability of a formula (with the assumption that numbers in graded modalities are represented in unary) [16].

4 Queries without Regular Expressions

In this section we study the problem of deciding whether $\mathcal{S} \models q \subseteq q'$, in the case where q does not contain regular expressions, i.e. atoms of the form $E(t,t')$. Note that the example in Section 2 falls into this case.

Our aim is to reduce query containment to a problem of unsatisfiability of 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

$$
\begin{array}{l}
(body_1(\vec{\mathbf{a}}, \vec{\mathbf{b}}_1, \vec{\mathbf{c}}_1) \vee \cdots \vee body_m(\vec{\mathbf{a}}, \vec{\mathbf{b}}_m, \vec{\mathbf{c}}_m)) \wedge \\
\neg\exists \vec{\mathbf{z}}_1 . body'_1(\vec{\mathbf{a}}, \vec{\mathbf{z}}_1, \vec{\mathbf{c}}_1) \wedge \cdots \wedge \neg\exists \vec{\mathbf{z}}_{m'} . body'_{m'}(\vec{\mathbf{a}}, \vec{\mathbf{z}}_{m'}, \vec{\mathbf{c}}_{m'})
\end{array}
$$

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

$$
\Phi = \Phi_{\mathcal{S}} \wedge (\bigvee_{j=1}^{m} \Phi_{body_j}) \wedge (\bigwedge_{j=1}^{m'} \Phi_{body'_j}) \wedge \Phi_{aux},
$$

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, and makes use of the mapping $\sigma(\cdot)$ from $\mathcal{DLR}_{reg}$ expressions to CPDL_g formulae defined in Figure 3. 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 Φ. Due to the connected-model property of CPDL_g, U 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 $\sigma(S)$, for all assertions $S \in \mathcal{S}$, in turn conjoined to the following formulae:

$$
\begin{array}{ll}
[U](\top_1 \vee \cdots \vee \top_{n_{max}}) & \\
[U]([f_i]_{\leq 1}\top) & \text{for each } i \in \{1, \ldots, n_{max}\} \\
[U]([f_i]\bot \supset [f_{i+1}]\bot) & \text{for each } i \in \{1, \ldots, n_{max}\} \\
[U](\top_n \equiv \langle f_1 \rangle \top_1 \wedge \cdots \wedge \langle f_n \rangle \top_1 \wedge [f_{n+1}]\bot) & \\
 & \text{for each } n \in \{2, \ldots, n_{max}\} \\
[U](\mathbf{P} \supset \top_n) & \text{for each atomic relation } \mathbf{P} \text{ of arity } n \\
[U](A \supset \top_1) & \text{for each atomic concept } A
\end{array}
$$

Intuitively, $\Phi_{\mathcal{S}}$ makes use of *reification* of n-ary relations, i.e. a tuple in a model of $\mathcal{S}$ is represented in a model of Φ by a state having one functional link f_i for each tuple component $\$i$.

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

For each $j \in \{1, \ldots, m\}$, the encoding Φ_{body_j} of $body_j(\vec{\mathbf{a}}, \vec{\mathbf{b}}_j, \vec{\mathbf{c}}_j)$ 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 $body_j(\vec{\mathbf{a}}, \vec{\mathbf{b}}_j, \vec{\mathbf{c}}_j)$. 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. $\vec{\mathbf{t}}$).

The formula Φ_{body_j} is the conjunction of the following formulae:

- for each $\mathbf{R}(\vec{\mathbf{t}})$ in $body_j(\vec{\mathbf{a}}, \vec{\mathbf{b}}_j, \vec{\mathbf{c}}_j)$

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

- for each $C(t)$ in $body_j(\vec{\mathbf{a}}, \vec{\mathbf{b}}_j, \vec{\mathbf{c}}_j)$

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

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

Figure 3: Mapping $\sigma(\cdot)$ from $\mathcal{DLR}_{reg}$ to CPDL$_g$

- for each $N_{\vec{\mathbf{t}}}$ with $\vec{\mathbf{t}} = (t_1, \ldots, t_n)$

  $$[U](N_{\vec{\mathbf{t}}} \equiv \langle f_1\rangle N_{t_1} \wedge \cdots \wedge \langle f_n\rangle N_{t_n} \wedge [f_{n+1}]\bot)$$

  and for each N_{t_i}

  $$[U](N_{t_i} \supset \langle f_i^-\rangle N_{\vec{\mathbf{t}}}) \qquad \text{and} \qquad [U](N_{t_i} \supset [f_i^-]_{\leq 1} N_{\vec{\mathbf{t}}})$$

Intuitively, Φ_{body_j} expresses the relationships between terms in $body_j(\vec{\mathbf{a}}, \vec{\mathbf{b}}_j, \vec{\mathbf{c}}_j)$ by using reification and name-formulae.

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

For each $j \in \{1, \ldots, m'\}$, $\Phi_{body'_j}$ encodes $\neg\exists\vec{\mathbf{z}}_j.body'_j(\vec{\mathbf{a}}, \vec{\mathbf{z}}_j, \vec{\mathbf{c}}_j)$ 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[3]. 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 $\vec{\mathbf{a}}$, $\vec{\mathbf{z}}_j$, $\vec{\mathbf{c}}_j$, and one node $\vec{\mathbf{t}}$ for each $\vec{\mathbf{t}}$ such that $\mathbf{R}(\vec{\mathbf{t}})$ appears in $\neg\exists\vec{\mathbf{z}}_j.body'_j(\vec{\mathbf{a}}, \vec{\mathbf{z}}_j, \vec{\mathbf{c}}_j)$. Each node t is labeled by N_t and all $\sigma(C)$ such that $C(t)$ appears in $\neg\exists\vec{\mathbf{z}}_j.body'_j(\vec{\mathbf{a}}, \vec{\mathbf{z}}_j, \vec{\mathbf{c}}_j)$. Each node $\vec{\mathbf{t}}$ is labeled by all $\sigma(\mathbf{R})$ such that $\mathbf{R}(\vec{\mathbf{t}})$ appears in $\neg\exists\vec{\mathbf{z}}_j.body'_j(\vec{\mathbf{a}}, \vec{\mathbf{z}}_j, \vec{\mathbf{c}}_j)$.

- 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 $\neg\exists\vec{\mathbf{z}}_j.body'_j(\vec{\mathbf{a}}, \vec{\mathbf{z}}_j, \vec{\mathbf{c}}_j)$. In addition, there is one edge labeled by $\sigma(E)$ from the node t to the node t', for each atom $E(t, t')$ occurring in $\neg\exists\vec{\mathbf{z}}_j.body'_j(\vec{\mathbf{a}}, \vec{\mathbf{z}}_j, \vec{\mathbf{c}}_j)$.

In general the tuple-graph is composed of $m \geq 1$ connected components. For the i-th connected component we build a CPDL$_g$ formula $\delta_i(\vec{\mathbf{z}}_j)$ by starting from a node t_0 (corresponding to a term) and visiting the corresponding component as follows (let u be the current node in the visit and ϕ_u the formula produced by visiting u):

- If $u = t$, and has not already been visited then construct ϕ_u as the conjunction of:

  (i) every formula labeling the node t (including N_t);

  (ii) one formula $\langle f_i^-\rangle\phi$ for each non-marked edge $(\vec{\mathbf{t}}, t)$ labeled by f_i (i.e. $t = t_i$ in $\vec{\mathbf{t}}$), where ϕ is the formula resulting by marking the edge $(\vec{\mathbf{t}}, t)$ and visiting the node $\vec{\mathbf{t}}$;

  (iii) one formula $\langle\sigma(E)\rangle(\phi \wedge \langle\sigma(E)^-\rangle N_t)$ for each non-marked edge (t, t') labeled by $\sigma(E)$, where ϕ is the formula resulting by marking the edge (t, t') and visiting the node t';

  (iv) one formula $\langle\sigma(E)^-\rangle(\phi \wedge \langle\sigma(E)\rangle N_t)$ for each non-marked edge (t', t) labeled by $\sigma(E)$, where ϕ is the formula resulting by marking the edge (t', t) and visiting the node t';

- If $u = \vec{\mathbf{t}} = (t_1, \ldots, t_n)$, let $e_1, \ldots, e_h$ be the non-marked edges from u to nodes t_i. Mark $e_1, \ldots, e_h$ and construct ϕ_u as the conjunction of:

  (i) every formula labeling the node $\vec{\mathbf{t}}$;

  (ii) one formula $\langle f_i\rangle\phi$ for each edge $e_j = (\vec{\mathbf{t}}, t_i)$, where f_i is the label of e_j, and ϕ is the formula resulting by visiting the node t_i.

- If u has already been visited then it corresponds to a term t, and the resulting formula ϕ_u is N_t.

Then $\delta_i(\vec{\mathbf{z}}_j) = \phi_{t_0}$. Observe that $\delta_i(\vec{\mathbf{z}}_j)$ contains newly introduced formulae N_{z_i}, one for each z_i in $\vec{\mathbf{z}}_j$.

$\Phi_{body'_j}$ consists of the conjunction of all formulae obtained by replacing in

$$([U]\neg\delta_1(\vec{\mathbf{z}}_j)) \vee \cdots \vee ([U]\neg\delta_\ell(\vec{\mathbf{z}}_j))$$

(i) each N_{z_i} not occurring in a cycle in the tuple-graph by $\top_1$, and

(ii) each N_{z_i} occurring in a cycle in the tuple-graph by each 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 or $\vec{\mathbf{c}}'_1, \ldots, \vec{\mathbf{c}}'_{m'}$ occurring in q'.

Observe that the number of such conjuncts in $\Phi_{body'_j}$ is $O(\ell_1^{\ell_2})$, where ℓ_1 is the number of variables and constants in q plus the number of constants in q', and ℓ_2 is the number of variables z_i occurring in a cycle in the tuple-graph for q'.

Φ_{aux}: encoding of constants and variables

Let $\Phi' = \Phi_{\mathcal{S}} \wedge (\bigvee_{j=1}^{m} \Phi_{body_j}) \wedge (\bigwedge_{j=1}^{m'} \Phi_{body'_j})$, and $N_1, \ldots, N_K$ all name-formulae in Φ'. Φ_{aux} is formed by the conjunction of:

[3]The tuple graph is similar to the graph used in [13] to detect cyclic dependencies between variables.

- the formula $\langle create\rangle N_1 \wedge \cdots \wedge \langle create\rangle N_K$ which expresses the existence of a state satisfying a name-formula N_i, for each $i \in \{1,\ldots,K\}$;
- one formula of the form $[U](N_{c_i} \supset \neg N_{c_j})$ for each pair of distinct constants c_i, c_j appearing in the queries (not Skolem constants);
- one formula of the form $[U](N_i \wedge \phi \supset [U](N_i \supset \phi))$ for each name-formula N_i, $i \in \{1,\ldots,K\}$, and each formula ϕ such that[4]:

  (a) $\phi \in CL(\Phi')$,

  (b) $\phi = \langle\overline{r}\rangle\phi'$ with $\langle r\rangle\phi' \in CL(\Phi')$, and

  (c) $\phi = \langle\overline{r}'; p\rangle N_j$ with $r' \in Pre(r)$, $p = f \mid f^-$, and r, f, N_j occurring in $CL(\Phi')$

  where $\overline{r}$ is defined inductively as follows:

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

The role of Φ_{aux} is to enforce that, in every model of Φ, for each N_k, one representative state can be singled out among those satisfying N_k. 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 polynomial with respect to Φ') described above. Theorem 1 proves that this is sufficient for our purposes.

To see how Φ encodes the containment problem, consider two queries $q(x_1, x_2) \leftarrow p(x_1, x_2)$, and $q'(x_1, x_2) \leftarrow r(x_1, x_2, z)$ over a schema $\mathcal{S}$ such that $\mathcal{S} \not\models q \subseteq q'$. Figure 4 schematically shows a model of the encoding Φ in this case, that represents a counterexample to the containment. Indeed, the model contains a state $s_{\vec{t}}$ in which p holds, that, being connected to s_{a_1}, s_{a_2} by means of f_1 and f_2, respectively, represents a tuple (a_1, a_2) that satisfies p. Since s_{a_1}, s_{a_2} satisfy N_{a_1}, N_{a_2}, respectively, and $\Phi_{body'} = [U](N_{a_1} \supset [f_1^-](r \supset [f_2](\neg N_{a_2} \vee [f_3]\bot)))$ is true in s_{root}, it follows that s_{a_1} satisfies $[f_1^-](r \supset [f_2]\neg N_{a_2})$. Therefore, in the model there is no state satisfying r representing a tuple (a_1, a_2, z).

By exploiting the properties of the encoding Φ, we can now prove decidability of query containment in our case.

Theorem 1 *Let $\mathcal{S}$ be a schema, q, q' be two queries, and let q not contain regular expressions. Then deciding whether $\mathcal{S} \models q \subseteq q'$ can be done in time $O(2^{p(|\mathcal{S}| \cdot \ell_1^{\ell_2})})$, where $|\mathcal{S}|$ is the size of $\mathcal{S}$, ℓ_1 is the sum of the number of variables in q and the number of constants in q and q', and ℓ_2 is the number of existentially quantified variables in q' that appear in a cycle of the tuple-graph for q'.*

Proof (sketch). **Soundness of the encoding**: Φ unsatisfiable implies $\mathcal{S} \models q \subseteq q'$. One can verify that every model $\mathcal{I}$ of $\mathcal{S}$ in which there is at least one tuple satisfying q and not q', can be turned into a model of Φ.

[4] $CL(\phi)$ is the Fisher-Ladner closure of a CPDL$_g$ formula ϕ, and $Pre(r)$ is the set of "prefixes" of a program r [16].

Completeness of the encoding: Φ satisfiable implies $\mathcal{S} \not\models q \subseteq q'$. We need to consider *tuple-admissible* models, i.e. models where there is no pair of states that represent the same reified tuple. We first prove that if Φ is satisfiable, then it admits a tuple-admissible model in which each name-formula is true in exactly one state. By the tree-model property, Φ admits a tree-model $\mathcal{M}' = (S', \cdot^{\mathcal{M}'})$, which is obviously 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 interprets name-formulae as singletons and is still tuple-admissible, as follows. For each N_i, $i \in \{1,\ldots,K\}$, we select a state s_{N_i}, among the states $s \in N_i^{\mathcal{M}'}$ such that $(s_{root}, s) \in create^{\mathcal{M}'}$. Then we define:

$$\begin{array}{l} create^{\mathcal{M}} = \{(s_{root}, s_{N_i}) \in create^{\mathcal{M}'} \mid i \in \{1,\ldots,K\}\} \\ p^{\mathcal{M}} = (p^{\mathcal{M}'} - \{(s_{N_i}, s) \in p^{\mathcal{M}'} \mid s \in N_j^{\mathcal{M}'}, i,j \in \{1,\ldots,K\}\} \\ \qquad\qquad - \{(s, s_{N_j}) \in p^{\mathcal{M}'} \mid s \in N_i^{\mathcal{M}'}, i,j \in \{1,\ldots,K\}\}) \\ \quad \cup \{(s_{N_i}, s_{N_j}) \mid (s_{N_i}, s) \in p^{\mathcal{M}'}, s \in N_j^{\mathcal{M}'}, i,j \in \{1,\ldots,K\}\} \\ \qquad\qquad\qquad \text{for each atomic program } p \text{ except } create \\ N_i^{\mathcal{M}} = \{s_{N_i}\} \qquad \text{for each name-formula } N_i,\ i \in \{1,\ldots,K\} \\ A^{\mathcal{M}} = A^{\mathcal{M}'} \cap S \\ \qquad\qquad \text{for each atomic formula } A \text{ except name formulae} \\ S = \{s_{root}\} \cup \\ \qquad \{s \in S \mid (s_{root}, s) \in create^{\mathcal{M}'} \circ (\bigcup_p (p^{\mathcal{M}'} \cup (p^-)^{\mathcal{M}'}))^*\} \end{array}$$

To show that $\mathcal{M}$ is indeed a model one can proceed as in [16].

We now prove that if Φ has a tuple-admissible model $\mathcal{M}$ in which each name-formula is true in exactly one state, then $\mathcal{S} \not\models q \subseteq q'$. We do so by showing how to construct from $\mathcal{M}$ a model $\mathcal{I}$ of $\mathcal{S}$ that makes the formula $(body_1(\vec{\mathbf{a}}, \vec{\mathbf{b}}_1, \vec{\mathbf{c}}_1) \vee \cdots \vee body_m(\vec{\mathbf{a}}, \vec{\mathbf{b}}_m, \vec{\mathbf{c}}_m)) \wedge \neg\exists\vec{\mathbf{z}}_1.body'_1(\vec{\mathbf{a}}, \vec{\mathbf{z}}_1, \vec{\mathbf{c}}_1) \wedge \cdots \wedge \neg\exists\vec{\mathbf{z}}_{m'}.body'_{m'}(\vec{\mathbf{a}}, \vec{\mathbf{z}}_{m'}, \vec{\mathbf{c}}_{m'})$ true. $\mathcal{I}$ is built as follows: $\Delta^{\mathcal{I}} = \top_1^{\mathcal{M}}$, $\mathbf{P}^{\mathcal{I}} = \{\vec{\mathbf{s}} \mid \exists s' \in \mathbf{P}^{\mathcal{M}}.(s', s_i) \in f_i^{\mathcal{M}}, i \in \{1,\ldots,n\}\}$, $A^{\mathcal{I}} = A^{\mathcal{M}}$, and $t^{\mathcal{I}} = s \in N_t^{\mathcal{M}}$ for each constant and Skolem constant t in q and q'. To show that $\mathcal{I}$ does the job, the most difficult part is to show that $\vec{\mathbf{a}} \notin q'^{\mathcal{I}}$, i.e., for one j $\neg\exists\vec{\mathbf{z}}_j.body'_j(\vec{\mathbf{a}}, \vec{\mathbf{z}}_j, \vec{\mathbf{c}}_j)$ is true in $\mathcal{I}$. Conceptually, we need to distinguish between two cases, depending on whether there is a cycle in the *tuple-graph* for $\neg\exists\vec{\mathbf{z}}_j.body'_j(\vec{\mathbf{a}}, \vec{\mathbf{z}}_j, \vec{\mathbf{c}}_j)$.

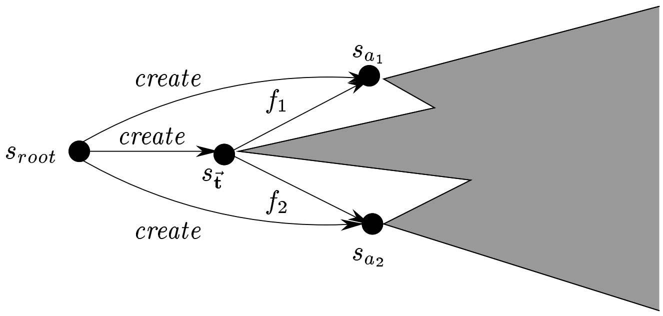
If there is no cycle in the tuple-graph, then the conjunct Φ_{body_j} directly enforces the constraints expressed by $\neg\exists\vec{\mathbf{z}}_j.body'_j(\vec{\mathbf{a}}, \vec{\mathbf{z}}_j, \vec{\mathbf{c}}_j)$.

If there is a cycle then, due to the fundamental inability of expressing in PDLs that two chains of links meet the same state, no CPDL$_g$ formula can directly express $\neg\exists\vec{\mathbf{z}}_j.body'_j(\vec{\mathbf{a}}, \vec{\mathbf{z}}_j, \vec{\mathbf{c}}_j)$. For the same reason, however, we can assume that the only cycles present in $\mathcal{M}$ are those formed by the states corresponding to the constants in the queries. Therefore the replacement of name-formulae in $\Phi_{body'_j}$ suffices.

Complexity: Since satisfiability in CPDL$_g$ is EXPTIME-complete, and the encoding is sound and complete, it follows that query containment can be done in time $O(2^{p(|\Phi|})$. It is easy to verify that $|\Phi| = O(|\mathcal{S}| \cdot \ell_1^{\ell_2})$. $\square$

5 Schemas and Queries without Number Restrictions

In this section we study the problem of deciding whether $\mathcal{S} \models q \subseteq q'$, where $\mathcal{S}$ is a schema and q, q' are two queries, in the case where neither the schema nor the queries contain



Φ	is true in	s_{root}
$N_{\vec{\mathbf{t}}} \wedge p$	is true in	$s_{\vec{\mathbf{t}}}$
N_{a_1}	is true in	s_{a_1}
N_{a_2}	is true in	s_{a_2}

Figure 4: A model of Φ

number restrictions. We consider an encoding Ψ defined as: $\Psi = \Phi_{\mathcal{S}} \wedge (\bigvee_{j=1}^{m} \Psi_{body_j}) \wedge (\bigwedge_{j=1}^{m'} \Phi_{body'_j}) \wedge \Psi_{aux}$, where $\Phi_{\mathcal{S}}$ and $\Phi_{body'_j}$ are defined as in the previous section, and Ψ_{body_j}, Ψ_{aux} are variants of Φ_{body_j}, Φ_{aux}, as described below.

The encoding Ψ_{body_j} of $body_j(\vec{\mathbf{a}}, \vec{\mathbf{b}}_j, \vec{\mathbf{c}}_j)$ makes again use of name-formulae, whose distinguishing properties are specified by Ψ_{aux}. Name-formulae are introduced for terms only (not for tuples). Specifically, one name-formula N_t is introduced for each term t in $\vec{\mathbf{a}}$, $\vec{\mathbf{b}}_j$, $\vec{\mathbf{c}}_j$. The formula Ψ_{body_j} is the conjunction of the following formulae:

- for each $\mathbf{R}(t_1, \ldots, t_n)$ in $body_j(\vec{\mathbf{a}}, \vec{\mathbf{b}}_j, \vec{\mathbf{c}}_j)$

  $$[U](\langle f_1 \rangle N_{t_1} \wedge \cdots \wedge \langle f_n \rangle N_{t_n} \wedge [f_{n+1}]\bot \ \supset \ \sigma(\mathbf{R}))$$

  and for each N_{t_i}

  $$[U](N_{t_i} \ \supset \ \langle f_i^- \rangle(\langle f_1 \rangle N_{t_1} \wedge \cdots \wedge \langle f_n \rangle N_{t_n} \wedge [f_{n+1}]\bot))$$

- for each $C(t)$ in $body_j(\vec{\mathbf{a}}, \vec{\mathbf{b}}_j, \vec{\mathbf{c}}_j)$

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

Let $\Psi' = \Phi_{\mathcal{S}} \wedge (\bigvee_{j=1}^{m} \Psi_{body_j}) \wedge (\bigwedge_{j=1}^{m'} \Phi_{body'_j})$, and let $N_1, \ldots, N_K$ be all name-formulae in Ψ'. The formula Ψ_{aux} is the conjunction of the following formulae:

- the formula $\langle create \rangle N_1 \wedge \cdots \wedge \langle create \rangle N_K$ where *create* is a newly introduced atomic program
- one formula of the form $[U](N_{c_i} \supset \neg N_{c_j})$ for each pair of distinct constants c_i, c_j appearing in the queries (not Skolem constants);
- one formula of the form $[U](N_i \wedge \phi \supset [U](N_i \supset \phi))$ for each name-formula N_i in Ψ' and each formula $\phi \in CL(\Psi')$.

We observe that the only graded modalities appearing in Ψ are those in $\Phi_{\mathcal{S}}$, imposing the functionality of all atomic programs[5].

By exploiting the properties of the encoding Ψ, we can now prove decidability of query containment in our case.

Theorem 2 *Let $\mathcal{S}$ be a schema, q, q' be two queries, and let S, q, q' not contain number restrictions. Then deciding whether $\mathcal{S} \models q \subseteq q'$ can be done in time $O(2^{p(|\mathcal{S}| \cdot \ell_1^{\ell_2})})$, where $|\mathcal{S}|$, ℓ_1, and ℓ_2 are as in Theorem 1.*

[5] This means that Ψ can be viewed as a Converse Deterministic PDL formula [35].

Proof (sketch). **Soundness of the encoding** and **complexity**: As in Theorem 1.

Completeness of the encoding: Ψ satisfiable implies $\mathcal{S} \not\models q \subseteq q'$. We first prove that if Ψ is satisfiable it admits a tuple-admissible model in which each name-formula is true in exactly one state. Let $\mathcal{M}' = (S', \cdot^{\mathcal{M}'})$ be a model of Ψ and $s_{root} \in \Psi^{\mathcal{M}'}$. By the results in [15] we can assume without loss of generality that $\mathcal{M}'$ is tuple-admissible. We transform $\mathcal{M}'$ into a new model $\mathcal{M} = (S, \cdot^{\mathcal{M}})$ with $S' \subseteq S$, which is still tuple-admissible and interprets name-formulae as singletons. For each N_i, $i \in \{1, \ldots, K\}$, we select a state s_{N_i}, among the states $s \in N_i^{\mathcal{M}'}$ such that $(s_{root}, s) \in create^{\mathcal{M}'}$. Then we define the model $\mathcal{M}$ as follows. For each name-formula N_i, $N_i^{\mathcal{M}} = \{s_{N_i}\}$. For each $s \in S' \subseteq S$, and for each atomic formula A which is not a name-formula, $s \in A^{\mathcal{M}}$ iff $s \in A^{\mathcal{M}'}$. For each $(s, s') \in p^{\mathcal{M}'}$ and for each atomic program p, $(s, s') \in p^{\mathcal{M}}$. In addition, for each $n \in \{1, \ldots, n_{max}\}$, and for each $s \in \top_n^{\mathcal{M}'}$ such that for some $s' \in \Delta^{\mathcal{M}'}$ (1) $(s, s') \in f_i^{\mathcal{M}'}$ for some $i \in \{1, \ldots, n\}$, and (2) $s' \in N_j^{\mathcal{M}'}$ for some $j \in \{1, \ldots, K\}$, we include in S a new state v and proceed as follows: (i) for every atomic formula A (including $\top_n$), $v \in A^{\mathcal{M}}$ iff $s \in A^{\mathcal{M}'}$; (ii) for $i \in \{1, \ldots, n\}$, let $(s, s_i) \in f_i^{\mathcal{M}'}$; if $s_i \in N_j^{\mathcal{M}'}$, for some $j \in \{1, \ldots, K\}$, then $(v, s_{N_j}) \in f_i^{\mathcal{M}}$, otherwise $(v, s_i) \in f_i^{\mathcal{M}}$. To show that $\mathcal{M}$ is indeed a model of Ψ we use the same technique as in [16], although in this case the proof is much simpler. Finally, we proceed as in Theorem 1 to prove that if Ψ has a tuple-admissible model $\mathcal{M}$ in which each name-formula is true in exactly one state, then $\mathcal{S} \not\models q \subseteq q'$. $\square$

6 Undecidability of Containment of Queries with Inequalities

In this section we show that if we allow for inequalities inside the queries, then query containment becomes undecidable. The proof of undecidability exploits a reduction from the unbounded *tiling problem* [5], which consists in deciding whether (a portion of) the integer grid can be tiled using a finite set of square tile types (fixed in orientation and with colored edges) in such a way that adjacent tiles have the same color on the common edge. As shown in [23], 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.

Theorem 3 *Let $\mathcal{S}$ be a schema, and q, q' two queries that may contain atoms of the form $t \neq t'$. Then the query containment problem $\mathcal{S} \models q \subseteq q'$ is undecidable.*

Proof (sketch). Consider an instance $\mathcal{T}$ of the tiling problem with tile types $D_1, \ldots, D_k$, where the colors on the four sides of tiles of type D_i are $left(D_i)$, $right(D_i)$, $up(D_i)$, and $down(D_i)$. 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:

$$
\begin{array}{rcl}
Tile & \sqsubseteq & (D_1 \sqcup \cdots \sqcup D_k) \sqcap \\
& & \exists[\$1](Right \sqcap (\$2 : Tile)) \sqcap \exists[\$1](Up \sqcap (\$2 : Tile)) \\
D_i & \sqsubseteq & Tile \qquad \text{for each } i \in \{1, \ldots, k\} \\
D_i & \sqsubseteq & \neg D_j \qquad \text{for each } i, j \in \{1, \ldots, k\},\ i \neq j \\
D_i & \sqsubseteq & (\bigvee_{j: left(D_j) = right(D_i)} \neg\exists[\$1](Right \sqcap (\$2 : \neg D_j))) \sqcap \\
& & (\bigvee_{j: down(D_j) = up(D_i)} \neg\exists[\$1](Up \sqcap (\$2 : \neg D_j)))
\end{array}
$$

Then there is a tiling of the upper right quadrant consistent with $\mathcal{T}$ iff $\mathcal{S}_\mathcal{T} \not\models q_0 \subseteq q'_0$, where:

$$
\begin{array}{rcl}
q_0() & \leftarrow & Tile(x) \\
q'_0() & \leftarrow & Right(x, y) \wedge Up(y, z) \wedge \\
& & Up(x, y') \wedge Right(y', z') \wedge z \neq z'.
\end{array}
$$

Indeed, from a tiling of the upper right quadrant consistent with $\mathcal{T}$ one obtains immediately a model of $\mathcal{S}_\mathcal{T}$ in which q_0 is true while q'_0 is false. Conversely, 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 the assertions in $\mathcal{S}_\mathcal{T}$ force the existence of arbitrary long chains of objects beginning with o_0 and connected one to the next by alternations of $Right^\mathcal{I}$ and $Up^\mathcal{I}$. Since q'_0 is false in $\mathcal{I}$, these chains of objects form indeed a grid. Moreover, by viewing such objects as tiles, the tiling constraints are satisfied due to the assertions in $\mathcal{S}_\mathcal{T}$. $\square$

The proof of Theorem 3 shows that query containment remains undecidable even in the restricted case where: (i) $\mathcal{S}$ does not contain assertions on relations, and all assertions on concepts are of the form $A \sqsubseteq C$, (ii) $\mathcal{S}$, q, and q' do not contain regular expressions or number restrictions, (iii) q and q' do not contain union, or constants expressions, and (iv) there is a single inequality in q', and no inequality in q.

7 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 the problem of checking query containment under the constraints expressed in the schema.

In particular, the decidability results refer to the case where either regular expressions are ruled out in the queries, or number restrictions or not allowed both in the schema and in the queries. The second case yields the first decidability result that we are aware of on containment of conjunctive queries with regular expressions. The decidability of query containment with regular expressions and number restrictions both in the schema and in the queries remains open, and will be the subject of future research.

Acknowledgments

This work was partly supported by MURST, ESPRIT LTR Project No. 22469 DWQ (Foundations of Data Warehouse Quality), the Italian Research Council (CNR) under Progetto Strategico "Informatica nella Pubblica Amministrazione", sottoprogetto PROGRESS (Reingegnerizzazione dei Processi e dei Dati nella Pubblica Amministrazione), and the Italian Space Agency (ASI) under project "Integrazione ed Accesso a Basi di Dati Eterogenee".

References

[1] Serge Abiteboul. Querying semi-structured data. In *Proc. of the 6th Int. Conf. on Database Theory (ICDT-97)*, pages 1–18, 1997.

[2] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases.* Addison Wesley Publ. Co., Reading, Massachussetts, 1995.

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

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

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

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

[7] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, and Riccardo Rosati. Description logic framework for information integration. In *Proc. of the 6th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-98)*, 1998.

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

[9] Edward P. F. Chan. Containment and minimization of positive conjunctive queries in oodb's. In *Proc. of the 11th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS-92)*, pages 202–211, 1992.

[10] Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proc. of the 5th ACM Sym. on Theory of Computing (STOC-77)*, pages 77–90, 1977.

[11] Ashok K. Chandra and Moshe Y. Vardi. The implication problem for functional and inclusion dependencies is undecidable. *SIAM J. on Computing*, 14(3):671–677, 1985.

[12] Surajit Chaudhuri and Moshe Y. Vardi. On the equivalence of recursive and nonrecursive Datalog programs. In *Proc. of the 11th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS-92)*, pages 55–66, 1992.

[13] Chandra Chekuri and Anand Rajaraman. Conjunctive query containment revisited. In *Proc. of the 6th Int. Conf. on Database Theory (ICDT-97)*, pages 56–70, 1997.

[14] Anthony C. Klug David S. Johnson. Testing containment of conjunctive queries under functional and inclusion dependencies. *J. of Computer and System Sciences*, 28(1):167–189, 1984.

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

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

[17] Guozhu Dong and Jianwen Su. Conjunctive query containment with respect to views and constraints. *Information Processing Letters*, 57(2):95–102, 1996.

[18] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Andrea Schaerf. AL-log: integrating datalog and description logics. *J. of Intelligent Information Systems*, 1998. To appear.

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

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

[21] A. Gupta and I. S. Mumick. Maintenance of materialized views: Problems, techniques, and applications. *IEEE Bulletin of the Technical Committee on Data Engineering*, 18(2):3–18, 1995.

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

[23] David Harel. Recurring dominoes: Making the highly undecidable highly understandable. *Annals of Discrete Mathematics*, 24:51–72, 1985.

[24] Richard Hull. Managing semantic heterogeneity in databases: A theoretical perspective. In *Proc. of the 16th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS-97)*, 1997.

[25] Yannis E. Ioannidis and Raghu Ramakrishnan. Containment of conjunctive queries: Beyond relations as sets. *ACM Trans. on Database Systems*, 20(3):288–324, 1995.

[26] Anthony C. Klug. On conjunctive queries containing inequalities. *J. of the ACM*, 35(1):146–160, 1988.

[27] Dexter Kozen and Jerzy Tiuryn. Logics of programs. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science – Formal Models and Semantics*, pages 789–840. Elsevier Science Publishers (North-Holland), Amsterdam, 1990.

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

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

[30] Alon Y. Levy and Dan Suciu. Deciding containment for queries with complex objects. In *Proc. of the 16th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS-97)*, pages 20–31, 1997.

[31] John C. Mitchell. The implication problem for functional and inclusion dependencies. *Information and Control*, 56:154–173, 1983.

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

[33] Jeffrey D. Ullman. Information integration using logical views. In *Proc. of the 6th Int. Conf. on Database Theory (ICDT-97)*, number 1186 in Lecture Notes in Computer Science, pages 19–40. Springer-Verlag, 1997.

[34] Ron van der Meyden. *The Complexity of Querying Indefinite Information*. PhD thesis, Rutgers University, 1992.

[35] Moshe Y. Vardi and Pierre Wolper. Automata-theoretic techniques for modal logics of programs. *J. of Computer and System Sciences*, 32:183–221, 1986. A preliminary version appeared in *Proc. of the 16th ACM SIGACT Symp. on Theory of Computing (STOC-84)*.

[36] Jennifer Widom. Special issue on materialized views and data warehousing. *IEEE Bulletin on Data Engineering*, 18(2), 1995.