CORE Metadata, citation and similar papers at core.ac.uk
Provided by UPCommons. Portal del coneixement obert de la UPC

Query Containment Checking as a View Updating Problem

(extended version)

Carles Farré
Ernest Teniente
Toni Urpí

Universitat Politècnica de Catalunya
LSI, Facultat d'Informàtica
Jordi Girona Salgado, 1-3
E-08034 Barcelona -- Catalonia

phone: +34 - 93 - 401 70 05
fax: +34 - 93 - 401 70 14
e-mail: [farre | teniente | urpi]@lsi.upc.es

Abstract

In this paper we present a new approach that handles query containment problems by expressing them as a view updating problem. Since this approach is independent of any particular view updating method, it provides a general framework that joins research efforts in both the query containment checking and view updating fields. In particular, the larger development of current view updating technology allows us to check properly query containment when considering negative-derived literals or integrity constraints. Existing methods for query containment checking that deal with these cases do not check actually containment but another related property called uniform containment, which is a sufficient but not necessary condition for containment. Therefore, an important outcome of our proposal is that, to the best of our knowledge, it is the first approach that checks "true" query containment instead of uniform query containment in the presence of negation and integrity constraints.

1. Introduction

Query Containment [Ull89] is the problem concerned with checking whether the answers that a query obtains against a database are a subset of the answers obtained by another query against the same database, independently of the specific content of the database. Query containment is applied in several contexts: query optimisation by removing redundant subexpressions [Ull89], materialised view and cache reuse [LMSS95], integrity constraints redundancy checking [GSUW94], etc.

An important amount of research has been devoted to query containment checking over the last 20 years [CM77, ASU79, JK83, Klu88, Sag88, Sar91, Ull89, CV92, LMSS93, LS93, ZO93, Gup94, LS95, DS96, ST96, CR97]. In general, the methods that have been proposed so far differ in several aspects such as the kind of databases they consider, the class of queries they can handle or the approach they take. However all of them seem to share a common base: they are intended to solve containment by first identifying and constraining themselves to a subclass of queries according to some syntactic conditions on the language the queries are expressed with, and then they propose an optimised solution using this syntactic knowledge. Following this approach, methods have been released which achieve polynomial-time performance by paying the price of covering rather small subclasses of queries.

Moreover, existing methods that deal with queries having negated-derived atoms in their rule bodies [LS93, ST96] do not check actually query containment but another related property called Uniform Containment [Sag88].

Consider, for instance, a deductive database consisting of two base predicates. *Emp*(x, d, s), indicates that employee x works at department d with a salary s. *Manager*(x, d) indicates that x is a manager of the department d. There are also two view predicates: *Boss*(x), which defines who is manager of some department, and *Chief*(x), defining who is manager and employee at the same department:

$$Boss(x) \leftarrow Manager(x, d)$$

$$Chief(x) \leftarrow Emp(x, d, s) \wedge Manager(x, d)$$

We define two queries, $\mathbf{Q}_1$ and $\mathbf{Q}_2$, with the same predicate query:

$$\mathbf{Q}_1\text{: } Subordinate(x) \leftarrow Emp(x, d, s) \wedge \neg Boss(x)$$

$$\mathbf{Q}_2\text{: } Subordinate(x) \leftarrow Emp(x, d, s) \wedge \neg Chief(x)$$

The intuition behind these queries is that the first query retrieves those employees that are not managers anywhere, while the second one retrieves those employees that are not employees and managers at some department at the same time. Clearly, the former query is more restrictive than the latter one, that is, all the answers that can be obtained by $\mathbf{Q}_1$ are also answers to $\mathbf{Q}_2$. Thus we say that $\mathbf{Q}_1$ is contained in $\mathbf{Q}_2$, written $\mathbf{Q}_1 \sqsubseteq \mathbf{Q}_2$. In contrast, it also seems clear that the symmetric case does not hold: the answers we can obtain with $\mathbf{Q}_2$ are not always a subset of the answers to $\mathbf{Q}_2$. Hence we say in this case that $\mathbf{Q}_2$ is not contained in $\mathbf{Q}_1$, written $\mathbf{Q}_2 \not\sqsubseteq \mathbf{Q}_1$.

This simple example without recursion nor built-in atoms cannot be solved in a satisfactory way by the methods proposed in [LS93, ST96]. These methods would check whether $\mathbf{Q}_1$ is uniform contained in $\mathbf{Q}_2$, written $\mathbf{Q}_1 \sqsubseteq^u \mathbf{Q}_2$. Uniform containment is a sufficient but not necessary condition for query containment: uniform containment implies containment, but the inverse is not true. In this example, [LS93, ST96] would prove that $\mathbf{Q_1}$ is not uniform contained in $\mathbf{Q_2}$[1], but this result serves neither to prove $\mathbf{Q_1} \sqsubseteq \mathbf{Q_2}$ nor $\mathbf{Q_1} \not\sqsubseteq \mathbf{Q_2}$. Moreover, since $\mathbf{Q_2} \not\sqsubseteq \mathbf{Q_1}$ is true in the example, it implies that $\mathbf{Q_2}$ cannot be uniform contained in $\mathbf{Q_1}$, written $\mathbf{Q_2} \not\sqsubseteq^u \mathbf{Q_1}$. Therefore, [LS93, ST96] would prove that $\mathbf{Q_2} \not\sqsubseteq^u \mathbf{Q_1}$, but again this result does not prove whether $\mathbf{Q_2} \sqsubseteq \mathbf{Q_1}$ is true or false.

When considering the integrity constraints defined in a database, the containment relationship between two queries does not need to hold for any state (content) of the database but only for the consistent ones, i.e. those that satisfy the integrity constraints. This idea is captured by the notion of **IC**-compliant Query Containment. Again, the current methods that deal with such an alternative containment concept [Sag88, ST96, DS96] take the uniform containment approach.

In this paper we present a new approach that reformulates the containment properties (both query and IC-compliant query containment) in terms of a view updating problem. Intuitively, the main idea of our approach is to express the accomplishment of the query containment relationship between two queries in terms of a derived (view) predicate. Then, checking containment is performed by requesting the insertion of that predicate. If some solution exists the predicate can be satisfied and, therefore, the expressed relationship holds.

This new approach is based on the seminal work of [DTU96], which uses view updating to perform database schema validation tasks, such as schema satisfiability or redundancy of integrity constraints. The reformulation of the containment problem we propose is similar to that of [LMSS93, LS95], which translates query containment to the problem of query satisfiability, i.e. a query is satisfiable if there is some database state in which the answer to the query is not empty. However, the query-satisfiability methods that are provided by [LMSS93, LS95] impose stronger restrictions on the cases that they handle, and they do not consider **IC**-compliant query containment.

The main contributions of our approach are:

- To the best of our knowledge, it is the first approach that tackles broadly the containment-checking problem for queries with negated-derived atoms in the rule bodies, by checking query containment instead of uniform query containment.
- As far as we know, it is the first approach that handles Query Containment with respect to integrity constraints, by checking query containment instead of uniform query containment.
- Our approach provides also a general framework that unifies research efforts in both the query containment checking and view updating fields. In our case, the larger development of the current view updating technology, with respect to the treatment of negation and integrity constraints, has allowed us to obtain new results for containment checking.
- Moreover, this framework is generic enough to be independent of any particular view updating method. In this way, any future advance in the view updating technology can be also applied in the query containment checking field.

This paper is organised as follows. Next section reviews basic concepts needed in the rest of the paper. Section 3 presents our approach to check query containment by view updating. Section 4 shows how to use view updating to check **IC**-compliant query containment. Section 5 reviews some aspects of current view updating technology. Section 6 discuses related work. Finally, section 7 presents our conclusions and points out further work.

[1] For more details, see section 6 below.

2. Base Concepts

In this section, we briefly review some definitions related to Deductive Databases, Queries, Query Containment and View Updating [Llo87, Ull88, Sag88, Ull89, TO95].

2.1 Deductive databases

Throughout the paper, we consider a first order language with a universe of constants -a, b, c, a_1, b_1, …-, a set of variables -*x*, *y*, *z*, x_1, y_1, …-, a set of predicate names -*P*, *Q*, *V*, P_1, Q_1, …- and no function symbols. A *term* is a variable or a constant. If *P* is a n-ary predicate and $t_1, \ldots, t_n$ are terms, then $P(t_1, \ldots, t_n)$ is an *atom*. The atom is *ground* if every $t_{i = 1,\ldots,n}$ is a constant. A *literal* is defined as either an atom or a negated atom.

A *fact* is a formula of the form: $P(t_1, \ldots, t_n) \leftarrow$, where $P(t_1, \ldots, t_n)$ is a ground atom. A *deductive rule* is a formula of the form:

$$P(t_1, \ldots, t_n) \leftarrow L_1 \wedge \ldots \wedge L_m \qquad \text{with } n \geq 0,\ m \geq 1$$

where $P(t_1, \ldots, t_n)$ is an atom denoting the *conclusion* and $L_1, \ldots, L_m$ are literals representing *conditions*. $P(t_1, \ldots, t_n)$ is called the *head* and $L_1 \wedge \ldots \wedge L_m$ the *body* of the deductive rule. Variables in the conclusion or in the conditions are assumed to be universally quantified over the whole formula. The definition of a predicate *P* is the set of all rules in the deductive database that have *P* in their head.

An *integrity constraint* is a formula that every state of the deductive database is required to satisfy. We deal with constraints in *denial* form:

$$\leftarrow L_1 \wedge \ldots \wedge L_m \qquad \text{with } m \geq 1$$

where the L_i are literals and all variables are assumed to be universally quantified over the whole formula. More general constraints can be transformed into this form by applying the procedure described in [LT84].

For the sake of uniformity, we associate to each integrity constraint an inconsistency predicate *Icn*, with or without terms, and thus they have the same form as the deductive rules. We call them *integrity rules*. Then, we would rewrite the former denial as $Ic1 \leftarrow L_1 \wedge \ldots \wedge L_m$. We also define an standard auxiliary predicate *Ic* with the following rules: $Ic \leftarrow Ic1, \ldots, Ic \leftarrow Icn$, one integrity rule for each integrity constraint of the database. A fact *Ic* will indicate that there is an integrity constraint that is violated.

A *deductive database* **D** is a triple **D** = (**EDB**, **DR**, **IC**) where **EDB** is a finite set of facts, **DR** a finite set of deductive rules, and **IC** a finite set of integrity constraints. The set **EDB** of facts is called the *extensional* part of the deductive database and the set **DR** of deductive rules is called the *intensional* part.

As usual, we also assume that deductive database predicates are partitioned into base and derived predicates. A base predicate appears only in the extensional part and (eventually) in the body of deductive rules. A derived (view) predicate appears only in the intensional part. Every deductive database can be defined in this form [BR86]. Furthermore, predicates in the body of deductive rules may be ordinary or evaluable ("built-in"). The former are base or view predicates, while the latter are predicates such as the comparison or arithmetic predicates, that can be evaluated without accessing the database.

A finite set of deductive rules (e.g. DR) is *recursive* if there is one ore more cycles in its *dependency graph*, which is built by drawing one edge between each pair of ordinary predicates appearing, respectively, in the head and the body of the same rule. As usual too, we do not admit any negative literal of a derived predicate defined direct or indirectly in a recursive way (*stratified negation*), and the variables appearing in negated atoms, as well as in evaluable ones, must also appear in a ordinary positive literal in the same rule body (*safeness*).

2.2 Queries, Query Containment and IC-compliant Query Containment

A *query* **Q** for a deductive database **D** is a finite set of deductive rules which defines a dedicated n-ary *query predicate* *Q*. We assume without loss of generality that the only derived predicate heading the deductive rules in **Q** is *Q* and that all predicates, base or derived, other than *Q* appearing in the bodies of the rules belong to **D**.

The *answer* to the query is the set of all ground facts about *Q* obtained as a result of evaluating the deductive rules from both **Q** and **DR** on **EDB**:

$$\{Q(a^i_1,\ldots,a^i_n) \mid Q(a^i_1,\ldots,a^i_n) \in (\mathbf{Q} \cup \mathbf{DR})(\mathbf{EDB})\}$$

where (**Q** ∪ **DR**)(**EDB**) is the smallest fixpoint of a bottom-up evaluation of the deductive rules in **Q** ∪ **DR** on **EDB**. Notice that, in particular, **Q** could be a subset of **DR**, e.g., a derived predicate defined in **DR** may also play the role of the query predicate.

A query $\mathbf{Q_1}$ is *contained* in an another query $\mathbf{Q_2}$ when the set of ground facts answering $\mathbf{Q_1}$ is a subset of the set of ground facts answering $\mathbf{Q_2}$, regardless of the underlying **EDB**.

Definition 2.1 Let $\mathbf{Q_1}$ and $\mathbf{Q_2}$ be two queries defining the same n-ary query predicate *Q* on a deductive database **D** = (**EDB**, **DR**, **IC**). $\mathbf{Q_1}$ *is contained in* $\mathbf{Q_2}$, written $\mathbf{Q_1} \sqsubseteq \mathbf{Q_2}$, if

$$\{Q(a^i_1,\ldots,a^i_n) \mid Q(a^i_1,\ldots,a^i_n) \in (\mathbf{Q_1} \cup \mathbf{DR})(\mathbf{EDB})\}$$
$$\subseteq \{Q(a^k_1,\ldots,a^k_n) \mid Q(a^k_1,\ldots,a^k_n) \in (\mathbf{Q_2} \cup \mathbf{DR})(\mathbf{EDB})\}$$

for any **EDB**. ◻

When considering integrity constraints, the containment relationship between two queries must not hold for any **EDB** but only for consistent **EDB**'s, i.e. those that satisfy the integrity constraints. As stated before, we assume that the database contains an inconsistency predicate *Ic* that holds whenever some integrity constraint is violated for a certain content of the **EDB**. Thus, consistent **EDB**'s are those where the fact *Ic* does not hold.

Definition 2.2 Let $\mathbf{Q_1}$ and $\mathbf{Q_2}$ be two queries defining the same n-ary query predicate *Q* on a deductive database **D** = (**EDB**, **DR**, **IC**). $\mathbf{Q_1}$ *is IC-compliant contained in* $\mathbf{Q_2}$, written $\mathbf{Q_1} \sqsubseteq_{\mathbf{IC}} \mathbf{Q_2}$, if

$$\{Q(a^i_1,\ldots,a^i_n) \mid Q(a^i_1,\ldots,a^i_n) \in (\mathbf{Q_1} \cup \mathbf{DR})(\mathbf{EDB})\}$$
$$\subseteq \{Q(a^k_1,\ldots,a^k_n) \mid Q(a^k_1,\ldots,a^k_n) \in (\mathbf{Q_2} \cup \mathbf{DR})(\mathbf{EDB})\}$$

for any **EDB** such that $Ic \notin (\mathbf{IC} \cup \mathbf{DR})(\mathbf{EDB})$. ◻

$Ic \notin (\mathbf{IC} \cup \mathbf{DR})(\mathbf{EDB})$ means that we do not obtain the *Ic* fact when applying the rules from both **IC** (integrity rules) and **DR** (deductive rules, since derived predicates may appear in the body of the constraints in **IC**) on **EDB**.

2.3 View Updating

For a deductive database **D** = (**EDB**, **DR**, **IC**), an *update* is a bipartite set **U** = **Ins** ∪ **Del** of ground facts about base predicates which defines a new deductive database **U**(**D**) = (**EDB'**, **DR**, **IC**) such that **EDB'** = (**EDB - Del**) ∪ **Ins**.

View updating is concerned with determining how a request to update a view (i.e. derived) fact should be appropriately translated into updates of the underlying base facts. For a deductive database **D** = (**EDB**, **DR**, **IC**) and a n-ary derived (view) predicate *V* defined in **DR**, a *view update request* is a formula of the form $\leftarrow$ *Insert*(*V*(a_1, …, a_n)) [or $\leftarrow$ *Delete*(*V*(a_1, …, a_n))]. An update **U** is a *translation* satisfying the view update request if the ground derived fact *V*(a_1, …, a_n) is evaluated true [or false] in **U**(**D**). A view update request *succeeds* when there is at least one translation, otherwise the view update request *fails*. In general, several translations may exist for a given view update request.

3. Query Containment Checking by View Updating

As we have seen before, a database extension (**EDB**) can be updated and changed into a new one by inserting or deleting base facts. Moreover, we can also request to update an **EDB** indirectly in terms of an insertion or a deletion of a view (derived) fact. In this later case, we need a view updating method that translates the view update request into insertions and/or deletions of base facts. Thus, view predicates with its defining deductive rules describe new properties and relationships over the underlying base predicates. Hence, we can see the request to insert some view fact as an intent to change the underlying **EDB** in order to satisfy and make true the property (or the relationship) expressed by that view fact.

Taking this into account, we may express the non-containment relationship between two queries in terms of a derived predicate. We do this by defining a new derived predicate, *Not_Cont_Q_1_in_Q_2*, such that its defining rule(s) expresses that the answers to the query $\mathbf{Q_1}$ are not a subset of the answers to $\mathbf{Q_2}$. When we request to insert the *Not_Cont_Q_1_in_Q_2* view fact on an **EDB** we indeed want to find whether there is some **EDB'** (reachable through some translation) on which $\mathbf{Q_1}$ is not contained in $\mathbf{Q_2}$. If there is some translation that satisfies the view update request $\leftarrow$ *Insert*(*Not_Cont_Q_1_in_Q_2*), then $\mathbf{Q_1}$ is not contained in $\mathbf{Q_2}$. Otherwise, $\mathbf{Q_1} \sqsubseteq \mathbf{Q_2}$.

We use view updating to check whether $\mathbf{Q_1}$ is not contained in $\mathbf{Q_2}$, written $\mathbf{Q_1} \not\sqsubseteq \mathbf{Q_2}$, since non-containment is proved by finding just one **EDB** on which the answers to $\mathbf{Q_1}$ are not a subset of the answers to $\mathbf{Q_2}$. Since containment is a property that needs to be fulfilled on every **EDB**, this non-containment test based on a view-updating formalism is nothing but a way to check containment (whether $\mathbf{Q_1} \sqsubseteq \mathbf{Q_2}$ holds) by a refuting prove.

Therefore, and in contrast with the preceding “syntactic” methods and proposals, our approach can be seen as a kind of simulation of how queries and the whole database behave, in the sense that view updating traces explicit and implicit relationships among rules in their intend to obtain translations leading to target database extensions, i.e. those on which the *Not_Cont_Q_1_in_Q_2* is true.

Lemma 3.1 Let $\mathbf{Q_1}$ and $\mathbf{Q_2}$ be two queries defining the same n-ary query predicate Q on a deductive database **D** = (**EDB**, **DR**, **IC**).

$\mathbf{Q_1} \sqsubseteq \mathbf{Q_2}$ iff the view update request $\leftarrow$ *Insert*($Not\_Cont\_Q_1\_in\_Q_2$) fails on **D'** = (Ø, **DR'**, Ø) where

Ø stands for both empty **EDB** and **IC**

$\mathbf{DR'} = \mathbf{DR} \cup \mathbf{Q_1'} \cup \mathbf{Q_2'} \cup \{Not\_Cont\_Q_1\_in\_Q_2 \leftarrow Q_1(x_1,\dots,x_n) \wedge \neg Q_2(x_1,\dots,x_n)\}$

$\mathbf{Q_1'}$ is the result of changing the query-predicate name Q for Q_1 in $\mathbf{Q_1}$

$\mathbf{Q_2'}$ is the result of changing the query-predicate name Q for Q_2 in $\mathbf{Q_2}$ ◻

As we said before, we propose a view-updating based approach to prove the lack of containment, i.e. whether there is an **EDB** such that

$$\{Q(a^i_1,\dots,a^i_n) \mid Q(a^i_1,\dots,a^i_n) \in (\mathbf{Q_1} \cup \mathbf{DR})(\mathbf{EDB})\}$$
$$\not\subset \{Q(a^k_1,\dots,a^k_n) \mid Q(a^k_1,\dots,a^k_n) \in (\mathbf{Q_2} \cup \mathbf{DR})(\mathbf{EDB})\}$$

which is equivalent to prove whether there is an **EDB** such that

$$\{<a^i_1,\dots,a^i_n> \mid Q_1(a^i_1,\dots,a^i_n) \in (\mathbf{Q_1'} \cup \mathbf{DR})(\mathbf{EDB})\}$$
$$\not\subset \{<a^k_1,\dots,a^k_n> \mid Q_2(a^k_1,\dots,a^k_n) \in (\mathbf{Q_2'} \cup \mathbf{DR})(\mathbf{EDB})\}$$

that is, whether we can obtain some tuple $<a^x_1,\dots,a^x_n>$ such that $Q_1(a^x_1,\dots,a^x_n)$ and $Q_2(a^x_1,\dots,a^x_n)$ are respectively true and false on some **EDB**.

Thus, the 0-ary derived predicate $Not\_Cont\_Q_1\_in\_Q_2$ that expresses the non-containment relationship is defined

$$Not\_Cont\_Q_1\_in\_Q_2 \leftarrow Q_1(x_1,\dots,x_n) \wedge \neg Q_2(x_1,\dots,x_n),$$

so that $\mathbf{Q_1} \not\sqsubseteq \mathbf{Q_2}$ iff $Not\_Cont\_Q_1\_in\_Q_2$ holds on some **EDB**.

The next step is to request the view update $\leftarrow$ *Insert*($Not\_Cont\_Q_1\_in\_Q_2$) on an arbitrary **EDB**. If the view update request succeeds using a concrete view updating method then it will mean that there is at least one translation[2] that leads to a new **EDB'** where $Not\_Cont\_Q_1\_in\_Q_2$ is evaluated true. Thus, it will be concluded that $\mathbf{Q_1} \not\sqsubseteq \mathbf{Q_2}$ is true from the definition of $Not\_Cont\_Q_1\_in\_Q_2$. Otherwise, it will be said that $\mathbf{Q_1} \sqsubseteq \mathbf{Q_2}$ is true. For simplicity and without loss of generality[3], the view insertion is requested on the empty **EDB**, i.e. the **EDB** that does not contain any base fact.

Note that we do not take into account the database integrity constraints -the **IC** set- to check query containment from the view updating approach, because the definition itself of containment requires such a property to be held on every **EDB** regardless of its consistency.

Example 3.1 Let us review the example we presented in the introduction, where we had a database **D** = (**EDB**, **DR**, **IC**) where

EDB is a set of ground facts about two base predicates *Emp*(*emp_name*, *department*, *salary*) and *Manager*(*emp_name*, *department*),

[2] Note that, in particular, the "empty" translation, i.e. the one with no insertions nor deletions of base facts, is also a valid result, meaning that the $Not\_Cont\_Q_1\_in\_Q_2$ fact is already true on the current **EDB**.
[3] We can reach any **EDB'** from any other **EDB** by doing the proper insertions and/or deletions of base facts on this **EDB**.

$$\begin{aligned}\mathbf{DR} = \{\ & \mathit{Boss}(x) \leftarrow \mathit{Manager}(x, d)\\ & \mathit{Chief}(x) \leftarrow \mathit{Emp}(x, d, s) \wedge \mathit{Manager}(x, d)\ \}\end{aligned}$$

and **IC** = Ø.

We define two queries with the same predicate query:

$\mathbf{Q_1}$: $\mathit{Subordinate}(x) \leftarrow \mathit{Emp}(x, d, s) \wedge \neg \mathit{Boss}(x)$

$\mathbf{Q_2}$: $\mathit{Subordinate}(x) \leftarrow \mathit{Emp}(x, d, s) \wedge \neg \mathit{Chief}(x)$

As we saw in the introduction, intuitively, $\mathbf{Q_1}$ is more restrictive than $\mathbf{Q_2}$, so we say that $\mathbf{Q_1}$ is contained in $\mathbf{Q_2}$. In order to check that $\mathbf{Q_1} \sqsubseteq \mathbf{Q_2}$ actually holds within our new framework, we first build the new database **D'** = (Ø, **DR'**, Ø) where

$$\begin{aligned}\mathbf{DR'} = \{ & \mathit{Not\_Cont\_Q_1\_in\_Q_2} \leftarrow \mathit{Subordinate_1}(x) \wedge \neg \mathit{Subordinate_2}(x)\\ & \mathit{Subordinate_1}(x) \leftarrow \mathit{Emp}(x, d, s) \wedge \neg \mathit{Boss}(x)\\ & \mathit{Subordinate_2}(x) \leftarrow \mathit{Emp}(x, d, s) \wedge \neg \mathit{Chief}(x)\\ & \mathit{Boss}(x) \leftarrow \mathit{Manager}(x, d)\\ & \mathit{Chief}(x) \leftarrow \mathit{Emp}(x, d, s) \wedge \mathit{Manager}(x, d)\ \}\end{aligned}$$

The next step is to perform the view update request $\leftarrow$ *Insert*(*Not_Cont_*Q_1*_in_*Q_2) on **D'** using a view updating method. In this case, such a request fails to succeed. Hence, $\mathbf{Q_1} \sqsubseteq \mathbf{Q_2}$ is true. Roughly, the main idea is that to satisfy $\mathit{Subordinate_1}(x)$ we have to insert $\mathit{Emp}(x, d, s)$. This insertion induces an insertion of $\mathit{Subordinate_2}(x)$, which can only be falsified by considering an additional insertion of $\mathit{Manager}(x, d)$. However, this new insertion falsifies at the same time $\mathit{Subordinate_1}(x)$. Therefore, it is not possible to find a translation leading to an **EDB** where $\mathit{Subordinate_1}(x)$ is true and $\mathit{Subordinate_2}(x)$ is false.

On the contrary, let us consider the opposite case: to check whether $\mathbf{Q_2} \sqsubseteq \mathbf{Q_1}$ holds. Again, we first need to build a new database **D''** = (Ø, **DR''**, Ø) where

$$\begin{aligned}\mathbf{DR''} = \{ & \mathit{Not\_Cont\_Q_2\_in\_Q_1} \leftarrow \mathit{Subordinate_2}(x) \wedge \neg \mathit{Subordinate_1}(x)\\ & \mathit{Subordinate_1}(x) \leftarrow \mathit{Emp}(x, d, s) \wedge \neg \mathit{Boss}(x)\\ & \mathit{Subordinate_2}(x) \leftarrow \mathit{Emp}(x, d, s) \wedge \neg \mathit{Chief}(x)\\ & \mathit{Boss}(x) \leftarrow \mathit{Manager}(x, d)\\ & \mathit{Chief}(x) \leftarrow \mathit{Emp}(x, d, s) \wedge \mathit{Manager}(x, d)\ \}\end{aligned}$$

Now, the view update request $\leftarrow$ *Insert*(*Not_Cont_*Q_2*_in_*Q_1) succeeds on **D''**. A possible translation might be: {*Insert*(*Emp*(joan, sales, 70000)), *Insert*(*Manager*(joan, accounting))}, so in the new **EDB** *Chief*(joan) is not true (joan is manager at a department other than the department where she is employee), but *Boss*(joan) is true. Therefore, we have found a translation that leads to an **EDB** where $\mathit{Subordinate_2}(x)$ is true but $\mathit{Subordinate_1}(x)$ is not. Thus we conclude that $\mathbf{Q_2} \not\sqsubseteq \mathbf{Q_1}$. □

4. IC-compliant Query Containment by View Updating

Due to the expressiveness of our approach, we can easily adapt it to take also into account the integrity constraints defined on the database. As we stated in section 2.2, **IC**-compliant query containment between two queries holds when the answers to one query are a subset of the answers obtained by the other query on any consistent **EDB**, i.e. on every **EDB** where no integrity constraint is violated. Translating this concept to our framework, we use view updating to check whether $\mathbf{Q_1}$ is not **IC**-compliant contained in $\mathbf{Q_2}$, written $\mathbf{Q_1} \not\sqsubseteq_{\mathbf{IC}} \mathbf{Q_2}$, since non-containment is proved by finding just one consistent **EDB** on which the answers to $\mathbf{Q_1}$ are not a subset of the answers to $\mathbf{Q_2}$. As we stated in section 2, a consistent **EDB** is that where the fact *Ic* does not hold, assuming that the database contains an inconsistency predicate *Ic* that holds whenever some integrity constraint is violated. So now we define a new derived predicate, $Not\_IC\_Cont\_Q_1\_in\_Q_2$, such that its defining rule(s) expresses that the answers to the query $\mathbf{Q_1}$ are not a subset of the answers to $\mathbf{Q_2}$, enforcing that *Ic* does not hold at the same time. When we request to insert the $Not\_IC\_Cont\_Q_1\_in\_Q_2$ view fact on an **EDB** we indeed want to find whether there is some **EDB'** (reachable through some translation) on which $\mathbf{Q_1}$ is not contained in $\mathbf{Q_2}$ and *Ic* is false. If there is some translation that satisfies the view update request $\leftarrow Insert(Not\_IC\_Cont\_Q_1\_in\_Q_2)$, then $\mathbf{Q_1}$ is not **IC**-compliant contained in $\mathbf{Q_2}$. Otherwise, $\mathbf{Q_1} \sqsubseteq_{\mathbf{IC}} \mathbf{Q_2}$.

Lemma 4.1 Let $\mathbf{Q_1}$ and $\mathbf{Q_2}$ be two queries defining the same n-ary query predicate *Q* on a deductive database **D** = (**EDB**, **DR**, **IC**).
$\mathbf{Q_1} \sqsubseteq_{\mathbf{IC}} \mathbf{Q_2}$ iff the view update request $\leftarrow Insert(Not\_IC\_Cont\_Q_1\_in\_Q_2)$ fails on **D'** = (Ø, **DR'**, **IC**), where

Ø stands for an empty **EDB**

$\mathbf{DR'} = \mathbf{DR} \cup \mathbf{Q_1'} \cup \mathbf{Q_2'} \cup$
$\{Not\_IC\_Cont\_Q_1\_in\_Q_2 \leftarrow Q_1(x_1,\ldots,x_n) \wedge \neg Q_2(x_1,\ldots,x_n) \wedge \neg Ic\}$

$\mathbf{Q_1'}$ is the result of changing the query-predicate name *Q* for Q_1 in $\mathbf{Q_1}$

$\mathbf{Q_2'}$ is the result of changing the query-predicate name *Q* for Q_2 in $\mathbf{Q_2}$ □

We use a view updating method to refute **IC**-query containment, i.e. to prove non- **IC**-query-containment, by finding an **EDB** satisfying both:

$$\{Q(a^i_1,\ldots,a^i_n) \mid Q(a^i_1,\ldots,a^i_n) \in (\mathbf{Q_1} \cup \mathbf{DR})(\mathbf{EDB})\}$$
$$\not\subset \{Q(a^k_1,\ldots,a^k_n) \mid Q(a^k_1,\ldots,a^k_n) \in (\mathbf{Q_2} \cup \mathbf{DR})(\mathbf{EDB})\} \text{ and}$$

$$Ic \notin (\mathbf{IC} \cup \mathbf{DR})(\mathbf{EDB})$$

which is equivalent to prove whether there is an **EDB** such that

$$\{<a^i_1,\ldots,a^i_n> \mid Q_1(a^i_1,\ldots,a^i_n) \in (\mathbf{Q_1'} \cup \mathbf{DR})(\mathbf{EDB})\}$$
$$\not\subset \{<a^k_1,\ldots,a^k_n> \mid Q_2(a^k_1,\ldots,a^k_n) \in (\mathbf{Q_2'} \cup \mathbf{DR})(\mathbf{EDB})\} \text{ and}$$

$$Ic \notin (\mathbf{IC} \cup \mathbf{DR})(\mathbf{EDB})$$

So we define the predicate $Not\_IC\_Cont\_Q_1\_in\_Q_2$ that expresses the non-**IC**-compliant-containment relationship as follows:

$$Not\_IC\_Cont\_Q_1\_in\_Q_2 \leftarrow Q_1(x_1,\ldots,x_n) \wedge \neg Q_2(x_1,\ldots,x_n) \wedge \neg Ic$$

meaning that the fact $Not\_IC\_Cont\_Q_1\_in\_Q_2$ will be true when there is a tuple $<a^x_1,\ldots,a^x_n>$ such that $Q_1(a^x_1,\ldots,a^x_n)$ is true and both $Q_2(a^x_1,\ldots,a^x_n)$ and *Ic* are false.

Again, we request $\leftarrow Insert(Not\_IC\_Cont\_Q_1\_in\_Q_2)$ on the empty **EDB** to check whether there is an **EDB** where $Not\_IC\_Cont\_Q_1\_in\_Q_2$ is evaluated true.

Example 4.1 Let us consider again the example 3.1 with

EDB is a set of ground facts about two base predicates *Emp*(*emp_name*, *department*, *salary*) and *Manager*(*emp_name*, *department*),

DR = { $Boss(x) \leftarrow Manager(x, d)$
$Chief(x) \leftarrow Emp(x, d, s) \wedge Manager(x, d)$ }

IC = { $Ic1 \leftarrow Emp(x, d1, s) \wedge Manager(x, d2) \wedge \mathrm{d1} \neq \mathrm{d2}$
$Ic \leftarrow Ic1$ }

And the two queries that we have been defined before:

$\mathbf{Q_1}$: $Subordinate(x) \leftarrow Emp(x, d, s) \wedge \neg Boss(x)$

$\mathbf{Q_2}$: $Subordinate(x) \leftarrow Emp(x, d, s) \wedge \neg Chief(x)$

So, we have introduced an integrity constraint, *Ic1*, forbidding any person to be employee and manager at different departments. If we do not take this integrity constraint into account, the query containment results we obtained in example 3.1 also hold, i.e. $\mathbf{Q_1} \sqsubseteq \mathbf{Q_2}$ and $\mathbf{Q_2} \not\sqsubseteq \mathbf{Q_1}$.

Since the answers to $\mathbf{Q_1}$ are always a subset of the answers obtained by $\mathbf{Q_2}$ on any **EDB** independently of its consistency, $\mathbf{Q_1} \sqsubseteq_{\mathbf{IC}} \mathbf{Q_2}$ also holds. On the contrary, although $\mathbf{Q_2} \not\sqsubseteq \mathbf{Q_1}$, we are going to demonstrate that $\mathbf{Q_2} \sqsubseteq_{\mathbf{IC}} \mathbf{Q_1}$ is true.

We check whether $\mathbf{Q_2} \sqsubseteq_{\mathbf{IC}} \mathbf{Q_1}$ holds by making the view update request $\leftarrow Insert(Not\_IC\_Cont\_Q_2\_in\_Q_1)$ on **D''** = (Ø, **DR''**, **IC**) where

DR'' = { $Not\_IC\_Cont\_Q_2\_in\_Q_1 \leftarrow Subordinate_2(x) \wedge \neg Subordinate_1(x) \wedge \neg Ic$
$Subordinate_1(x) \leftarrow Emp(x, d, s) \wedge \neg Boss(x)$
$Subordinate_2(x) \leftarrow Emp(x, d, s) \wedge \neg Chief(x)$
$Boss(x) \leftarrow Manager(x, d)$
$Chief(x) \leftarrow Emp(x, d, s) \wedge Manager(x, d)$ }

In this case the view update request fails and, then, $\mathbf{Q_2} \sqsubseteq_{\mathbf{IC}} \mathbf{Q_1}$ is true. Roughly, the fact is that all possible translations obtained before when checking $\mathbf{Q_2} \sqsubseteq \mathbf{Q_1}$, {*Insert*(*Emp*(joan, sales, 70000)), *Insert*(*Manager*(joan, accounting))} for instance, are not valid translations now because all of them require somebody to be manager of a department other that the department where she or he is assigned as employee, and this requirement violates necessarily the integrity constraint *Ic1*. □

5. View Updating Methods for Containment Checking

A wealth of methods have been proposed in the view updating research area [KM90, Wüt93, TO95, CHM95, Dec96, LT97, CTU97], just for mentioning some of them. These methods differ in several aspects such as the kind of databases considered, the type of updates they can handle or the approach taken to deal with the problem.

Our approach to handle query containment checking by using view updating is independent of the particular method used to perform the requested view update. However, a method able to deal with view updates in a certain class of deductive databases should satisfy the following requirements in order to be used to check query containment:

1. The class of deductive databases considered by the method must allow, at least, to express the declarative definitions of the non-containment relationships that we have defined in sections 3 and 4.
2. If there exists some translation that satisfies a given request, the method obtains one such translation, but not necessarily several or even all of them. Otherwise, if no translation exists, the method must terminate.

By translating containment checking in terms of a view updating problem, the potential of our proposal relies on the capabilities of current view updating technology. The definition of the non-containment predicates we provide already implies the negation of the containing query in the rule body. Therefore the view updating methods we choose, as stated in point 1 above, should at least be capable of performing view updates on derived predicates defined in such a way. The Events Method [TO95], for instance, is a view updating method that fits well in these requirements. All the examples we show in this paper are performed successfully by this method and, thus, the containment problems they pose are solved in a satisfactory way.

Our independence of particular methods also allows us to take advantage of the future advances of view updating technology. At this point, we want just to mention a new approach, [CTU97], which is aimed at obtaining intensional translations for view updating requests. Intuitively, an intensional translation characterises a set of possible values to be considered for the obtained base fact updates, instead of enumerate all of them. This promising new technology can impact favourably in our approach, in the sense that such intensional translations are more useful and meaningful for the purposes of query containment.

6. Related Work

In general, existing containment checking methods are intended to solve containment problems into well-defined subclasses of queries and databases, restricted according to some syntactic conditions on the language that expresses them. For example, most research has been concerned with containment checking of conjunctive queries [CM77, ASU79, JK83, Klu88, Ull89, Sar91, ZO93, Gup94, CR97] and different results are obtained according to the syntactic features they considered. However, our aims have not been addressed to provide more efficient algorithms for these cases, but to set up a widespread framework where we can handle as many cases as possible, including those cases that had never been dealt properly before.

The main point is that our approach has allowed us to tackle containment in the presence of negation and integrity constraints in a proper way. So in the next (sub-)sections we review the related work according to these two topics.

6.1 Query Containment with Safe Stratified Negation.

In this section we show how our approach handles queries with (safe stratified) negated-derived literals and (safe) built-in atoms, i.e. covering entirely all the features of our data model as we have defined it in section 2. We want to point out that our approach keeps checking “true” containment. We remark the word “true” to refer to the concept of containment such as we have dealt with it in the previous sections and it was defined in section 2, in contrast to the concept of uniform containment that we will review below. To the best of our knowledge, our approach is the first one that tackles broadly the containment-checking problem for datalog queries with negated-derived atoms (and built-in atoms) in their rule bodies, without taking a uniform-containment approach.

Since we express the noncontainment relationship between two queries by defining a view predicate with a deductive rule of the form $Not\_Cont\_Q_1\_in\_Q_2 \leftarrow Q_1(x_1,\ldots,x_n) \wedge \neg Q_2(x_1,\ldots,x_n)$, dealing with stratified negation is always a necessary requirement for our approach and for the view updating methods we can use, even when no rule in $\mathbf{Q_1}$, $\mathbf{Q_2}$ or **DR** has negated derived atoms.

As usual, safe stratified negation was first tackled inside well-defined boundaries: [LMSS93] proves that containment for queries with only 1-ary base predicates and stratified negation is decidable; and [LS95] provides an algorithm to check predicate satisfiability that can also be used to check containment of a datalog query, i.e. without negation, in a union of conjunctive queries having local negated-base and inequality atoms (a negated-base or inequality atom is local when all its variables appear in at least one base atom). However, "true" containment has not gone beyond these restrictive bounds until now.

In contrast, other research works tackle broadly datalog extensions from the "uniform-containment" point of view: [LS93] provides an algorithm to check Uniform Query Equivalence (i.e. whether both $\mathbf{Q_1} \sqsubseteq^u \mathbf{Q_2}$ and $\mathbf{Q_2} \sqsubseteq^u \mathbf{Q_1}$ hold) for datalog queries with stratified negation and safe built-in atoms; and [ST96] propose a more efficient but incomplete algorithm to perform Uniform Query Containment checking for queries with stratified negation.

Uniform query containment was coined in [Sag88] as an alternative concept to query containment and proved it to be decidable for Datalog queries, even when both of them are recursive. Let $\mathbf{Q_1}$ and $\mathbf{Q_2}$ be two queries defining the same n-ary query predicate Q on a deductive database $\mathbf{D} = (\mathbf{EDB}, \mathbf{DR}, \mathbf{IC})$. $\mathbf{Q_1}$ *is uniform contained in* $\mathbf{Q_2}$, written $\mathbf{Q_1} \sqsubseteq^u \mathbf{Q_2}$, if

$$\{Q(a^i_1,\ldots,a^i_n) \mid Q(a^i_1,\ldots,a^i_n) \in (\mathbf{Q_1} \cup \mathbf{DR})(\mathbf{I})\}$$
$$\subseteq \{Q(a^k_1,\ldots,a^k_n) \mid Q(a^k_1,\ldots,a^k_n) \in (\mathbf{Q_2} \cup \mathbf{DR})(\mathbf{I})\}$$

for every **I** being an arbitrary set of ground facts about base and derived (query or view) predicates. Note that derived facts in **I** are independent from and may not be related to the ones computed by applying the rules in **DR** (and/or the ones from the queries) on the base facts only.

As pointed out in [Sag88], Uniform Query Containment provides a sufficient but not necessary condition for Query Containment: $\mathbf{Q_1} \sqsubseteq^u \mathbf{Q_2} \Rightarrow \mathbf{Q_1} \sqsubseteq \mathbf{Q_2}$. Hence if the Uniform Query Containment test fails, i.e. $\mathbf{Q_1} \not\sqsubseteq^u \mathbf{Q_2}$, nothing can be said about whether $\mathbf{Q_1} \sqsubseteq \mathbf{Q_2}$ holds.

Although at first sight it seems more complex to take into account every possible combination of base and derived facts, indeed the model-theoretic based algorithm to test Uniform Query Containment for datalog queries is quite simple for the datalog case.

Example 6.1 Let us review again our working example. As we proved in the example 3.1, $\mathbf{Q_1} \sqsubseteq \mathbf{Q_2}$ holds because no translation exists for the view update request $\leftarrow Insert(Not\_Cont\_Q_1\_in\_Q_2)$ that we performed on the database **D'** such as we defined them according to our approach.

However, this simple example without recursion nor built-in atoms does not fall into the query subclasses that handle [LMSS93] and [LS95], so their proposals cannot be applied to this example.

On the other hand, any uniform-containment based method, either [LS93] or [ST96], would have to demonstrate that $\mathbf{Q_1} \sqsubseteq^u \mathbf{Q_2}$ holds in order to prove that $\mathbf{Q_1} \sqsubseteq \mathbf{Q_2}$ is true. However, the fact is that $\mathbf{Q_1} \sqsubseteq^u \mathbf{Q_2}$ does not hold, since it can be found an **I** such that $(\mathbf{Q_1} \cup \mathbf{DR})(\mathbf{I}) \not\subset (\mathbf{Q_2} \cup$

DR)(**I**). For example, let us consider **I** = { *Emp*(rose, sales, 9000), *Chief*(rose) }, according to the definition of uniform containment that allows **I** to contain also ground facts about derived predicates. Computing the answers for each query on **I** we obtain[4]:

- ($\mathbf{Q_1} \cup$ **DR**)(**I**) = { *Emp*(rose, sales, 9000), *Chief*(rose), *Subordinate*(rose) }, from applying

  DR = {$Boss(x) \leftarrow Manager(x, d)$
  $Chief(x) \leftarrow Emp(x, d, s) \wedge Manager(x, d)$ }

  and $\mathbf{Q_1}$: $Subordinate(x) \leftarrow Emp(x, d, s) \wedge \neg Boss(x)$

  so the answer to $\mathbf{Q_1}$ on **I** is *Subordinate*(rose). Note that the single rule from $\mathbf{Q_1}$ produces the fact *Subordinate*(rose) because *Boss*(rose) does not appear in **I**.

- ($\mathbf{Q_2} \cup$ **DR**)(**I**) = { *Emp*(rose, sales, 9000), *Chief*(rose) }, from applying

  DR = {$Boss(x) \leftarrow Manager(x, d)$
  $Chief(x) \leftarrow Emp(x, d, s) \wedge Manager(x, d)$ }

  and $\mathbf{Q_2}$: $Subordinate(x) \leftarrow Emp(x, d, s) \wedge \neg Chief(x)$

  so the answer to $\mathbf{Q_2}$ on **I** is Ø. Note that here the fact *Chief*(rose) in **I** does not allow the query rule from $\mathbf{Q_2}$ to produce *Subordinate*(rose).

Therefore any uniform-containment based method would fail to prove that $\mathbf{Q_1} \sqsubseteq^u \mathbf{Q_2}$ and, thus, it would not be able to show that $\mathbf{Q_1} \sqsubseteq \mathbf{Q_2}$ in this example. On the contrary, we have shown how our approach deals successfully with this case in the example 3.1. □

6.2 IC-compliant Query Containment

Integrity constraints as the so called tuple generating dependencies (TGD's) were already considered in [Sag88] to check **IC**-compliant query containment for datalog queries. Moreover, [ST96] extends [Sag88] by taking also equality generating dependencies into account and [DS96] provides a method to check **IC**-compliant query containment for conjunctive queries and disjunctive-datalog integrity rules. However, all those proposals tackle the problem from the uniform-containment approach, i.e. they do not check **IC**-compliant query containment but "**IC**-compliant uniform query containment". In section 6.1 we have already discussed the "weaknesses" of the uniform-containment approach, so our major contribution is that our proposal checks actually **IC**-compliant query containment in contrast to the other related work.

7. Conclusions and Further Work

We have presented a new approach for checking query containment that is based on the use of view updating. The idea is to define a view expressing the non-containment relationship between two queries. Then, a request to insert that view allows us to determine whether the non-containment relationship holds.

[4] Note that it is not an example that shows how a particular uniform-containment based method works, but it proves that $\mathbf{Q_1} \sqsubseteq^u \mathbf{Q_2}$ does not hold by using a counter-example following the definition of uniform containment.

Our approach is independent of any particular method for view updating and, in this way, we can take advantage of the current and future advances in the view updating research area. Moreover, it handles in a uniform way all query subclasses that have been identified up to now in the literature of deductive databases. Hence, we set up a new framework that unifies both containment checking and view updating research areas.

The application of the advances of the current view updating technology in this framework yields to another main contribution of our approach, which is that of checking properly "true" query containment in the presence of negation and integrity constraints for the first time. Existing methods either cannot cover broadly these cases or they do not check query containment but uniform query containment.

As further work, we are going to implement a new view updating method according to the approach proposed in [CTU97], in order to use it efficiently for containment checking. Hopefully, the intensional orientation of this method will also improve the meaningfulness of the containment checking tests and extend our framework to the field of constraint databases.

Other possible extensions of our work would be to consider query containment in the presence of aggregate functions, queries over bags, or in object oriented databases as addressed in [LS97, CV93, BH97, BJNS94], to mention some previous work.

Acknowledgements

This work has been partially supported by PRONTIC CICYT program project TIC97-1157.

References

[ASU79] A.V. Aho, Y. Sagiv, J.D. Ullman: "Efficient Optimization of a Class of Relational Expressions". *ACM Transactions on Database Systems*, Vol. 4, No. 4, 1979, pp. 435-454.

[BH97] N.R. Brisaboa, H.J. Hernández: "Testing Bag-Containment of Conjunctive Queries". *Acta Informatica*, Vol. 34, No.7, 1997, pp. 557-578.

[BJNS94] M. Buchheit, M.A. Jeusfeld, W. Nutt, M. Staudt: "Subsumption of queries in object-oriented databases". *Information Systems*, Vol. 19, No. 1, 1994, pp. 33-54.

[BR86] F. Bancilhon, R. Ramakrishnan: "An amateur's introduction to recursive query processing strategies". *Proceedings of the 1986 ACM SIGMOD International Conference on Management of Data. SIGMOD Record*, Vol. 15, No. 2, 1986, pp. 16-52.

[CHM95] I.A. Chen, R. Hull, D. McLeod: "An Execution Model for Limited Ambiguity Rules and Its Application to Derived Data Update". *ACM Transactions on Database Systems*,, Vol. 20, No. 4, 1995, pp. 365-413.

[CM77] A.K. Chandra, P.M. Merlin: "Optimal Implementation of Conjunctive Queries in Relational Data Bases", *Proceedings of the 9th ACM SIGACT Symposium on Theory of Computing*. 1977, pp. 77-90.

[CR97] C. Chekuri, A. Rajaraman: "Conjunctive Query Containment Revisited". *Proceedings of the 6th International Conference on Database Theory (ICDT'97)*. Lecture Notes in Computer Science, Vol. 1186, Springer, 1997, pp. 56-70.

[CTU97] D. Costal, E. Teniente, T. Urpí: "An Approach to Obtain Intensional Translations for Consistent View Updating". *Proocedings of the 5th International Conference on Deductive and Object-Oriented Databases (DOOD'97)*. Montreux, Switzerland, 1997, pp. 175-192.

[CV92] S. Chaudhuri, M.Y. Vardi: "On the Equivalence of Recursive and Nonrecursive Datalog Programs", *Proceedings of the 11th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PoDS'92)*. ACM Press, 1992, pp. 55-66.

[CV93] S. Chaudhuri, M. Vardi: “Optimizing real conjunctive queries”. *Proceedings of the 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PoDS’93)*. ACM Press, 1993, pp. 59-70.

[DS96] G. Dong, J. Su: “Conjunctive query containment with respect to views and constraints”. *Information Processing Letters*, No. 57, pp. 95-102, 1996.

[DTU96] H. Decker, E. Teniente, T. Urpí: “How to tackle schema validation by view updating”, *Proceedings of the 5th International Conference on Extending Database Technology (EDBT’96)*. Lecture Notes in Computer Science, Vol. 1057, Springer, 1996, pp. 535-549.

[GSUW94] A. Gupta, Y. Sagiv, J.D. Ullman, J. Widom: “Constraint Checking with Partial Information”, *Proceedings of the 13th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database System (PoDS’94)*. ACM Press, 1994 *s*, pp. 45-55.

[Gup94] A. Gupta: *Partial Information Based Integrity Constraint Checking*. PhD. Thesis, Stanford University, 1994.

[JK83] D.S. Johnson, A. Klug: “Optimizing conjunctive queries that contain untyped variables”. *SIAM Journal on Conputing*, Vol. 12, No. 4, pp. 616-640, 1983.

[Klu88] A. Klug: “On Conjunctive Queries Containing Inequalities”. *Journal of the ACM*, Vol. 35, No. 1, 1988, pp. 146-160.

[KM90] A. Kakas, P. Mancarella: “Database Updates through Abduction”, *Proceedings of the 16th Very Large Data Bases Conference (VLDB’90)*. Morgan Kaufmann, 1990, pp. 650-661.

[Llo87] J.W. Lloyd: *Foundations of Logic Programming*, Springer, 1987.

[LMSS93] A. Levy, I.S. Mumick, Y. Sagiv, O. Shmueli: “Equivalence, query-reachability and satisfiability in Datalog extensions”. *Proceedings of the 12th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PoDS’93)*. ACM Press, 1993, pp. 109-122.

[LMSS95] A. Levy, A. Mendelzon, Y. Sagiv, D. Srivastava: “Answering Queries Using Views”. *Proceedings of the 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PoDS’95)*. ACM Press, 1995, pp. 95-104.

[LS93] A. Levy, Y. Sagiv: “Queries Independent of Updates”, *Proceedings of the 19th Very Large Data Bases Conference (VLDB’93)*. Morgan Kaufmann, 1995, pp. 171-181.

[LS95] A. Levy, Y. Sagiv: “Semantic Query Optimization in Datalog Programs”. *Proceedings of the 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PoDS’95)*. ACM Press, 1995, pp. 163-173.

[LS97] A. Levy, D. Suciu: “Deciding Containment for Queries with Complex Objects”. *Proceedings of the 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PoDS’97)*. ACM Press, 1995, pp. 20-31

[LT84] J.W. Lloyd, R.W. Topor: “Making Prolog More Expressive”. *Journal of Logic Programming*, 1984, No. 3, pp. 225-240.

[LT97] J. Lobo, G. Trajcevski: “Minimal and consistent evolution of knowledge bases”. *Journal of Applied Non-Classical Logics*. Vol. 7, No. 1-2, 1997, pp. 117-146.

[Oli91] A. Olivé: “Integrity Checking in Deductive Databases”. *Proceedings of the 17th Very Large Data Bases Conference (VLDB’91)*. Morgan Kaufmann, 1991, pp. 513-523.

[Sag88] Y. Sagiv: “Optimizing Datalog Programs”. In J. Minker (Ed.): *Foundations of Deductive Databases and Logic Programming*. Morgan Kaufmann, Los Altos, CA, 1988, pp. 659-698.

[Sar91] Y. Saraiya: *Subtree elimination algorithms in deductive databases*. PhD. Thesis, Stanford University, 1991.

[ST96] M. Staudt, K.v. Thadden: “A Generic Subsumption Testing Toolkit for Knowledge Base Queries”. *Proceedings of the 7th International Conference on Database and Expert Systems Applications (DEXA’96)*. Lecture Notes in Computer Science, Vol. 1134, Springer, 1996, pp. 834-844.

[TO95] E. Teniente, A. Olivé: “Updating Knowledge Bases while Maintaining their Consistency”. *The VLDB Journal*, Vol. 4, No. 2, 1995, 193-241.

[Ull88] J.D. Ullman: *Principles of Database an Knowledge-Base Systems, Volume 1.* Computer Science Press, Rockville, MD, 1988.

[Ull89] J.D. Ullman: *Principles of Database an Knowledge-Base Systems, Volume 2: The New Technologies*. Computer Science Press, Rockville, MD, 1989.

[Wüt93] B. Wüthrich: “On Updates and Inconsistency Repairing in Deductive databases”. *Proceedings of the International Conference on Data Engineering (ICDE’93)*. IEEE Computer Society Press, 1993, pp. 608 - 615.

[ZO93] X. Zhang, M.Z. Ozsoyoglu: “On efficient reasoning with implication constraints”. *Proocedings of the 3th International Conference on Deductive and Object-Oriented Databases (DOOD’93)*. Lecture Notes in Computer Science, Vol. 760, Springer, 1993, pp. 236-252.