A general procedure to test containment of conjunctive queries

Miguel R. Penabad, Nieves R. Brisaboa, José R. Paramá
Departamento de Computación,
Universidade da Coruña
Spain

{penabad,brisaboa,parama}@udc.es

Hendrik Decker
PMS, Institut für Informatik
University of München
Germany

hdecker@informatik.uni-muenchen.de

Abstract

Equivalence and containment of queries is a crucial issue of database query optimization theory. Among the different perspectives used to study this problem, we consider in this work two orthogonal ones: the presence of inequalities in the queries, and the underlying semantics, that can be set or bag theoretic. This work presents a general procedure, QCC (Query containment Checker), that checks the containment of queries with or without inequalities, under set or bag semantics.

1 Introduction

Equivalence and containment of queries is a crucial issue of database query optimization theory. Containment in general is undecidable [12], but was shown to be decidable for the standard class of conjunctive queries by Chandra and Merlin [6]. In the latter, the only comparison operator allowed to occur in queries is equality. Subsequent studies also considered inequality queries, i.e., conjunctive queries with inequalities allowed, i.e., built-in predicates of the form $X\theta Y$, where $\theta \in \{=, \neq, <, \leq, >, \geq\}$. Klug [10] solved the containment problem for inequality queries. However, his solution works only for dense domains, but not for constants taking values from any nondense domain, e.g., the integers. Other papers related to query containment are [14, 9]. They show the decidability of problems that are equivalent to the containment of inequality queries, but do not offer any constructive procedure for testing containment. In [4], we presented a necessary and sufficient condition, along with a procedure, to test containment of inequality queries.

Most of the studies about query containment, including those cited above, assume a set-theoretic framework, as usual in the relational model. However, commercial database management systems use a bag (also called multiset) semantics. Bag semantics allows for multiple occurences of identical tuples in a database table. Likewise, any SQL query in a commercial DBMS produces a bag of tuples, unless the `Distinct` clause is used for removing duplicates. The use of a different collection semantics has strong implications for query containment. In particular, results achieved for set containment are not valid for bag containment. Chaudhuri and Vardi [7] gave partial solutions of the bag containment problem for equality queries; Brisaboa and Hernández specified an exact condition, along with a decision procedure. They solved bag containment testing of equality queries by transforming it into the problem of comparing pairs of polynomials over the set of nonnegative integers. However, there have yet been no publications, as far as we know, about effectively testing bag containment of inequality queries.

In this paper, we consider two orthogonal aspects of conjunctive queries: the presence or absence of inequalities, i.e., equality or inequality queries, on one hand, and the underlying set or bag semantics, on the other. According to these alternatives, query containment can be studied under four different perspectives:

- Set containment of equality queries;
- Set containment of inquality queries;
- Bag containment of equality queries;
- Bag containment of inquality queries.

We are going to present a general procedure to test conjunctive query containment for equality and inequality queries, which works under set and bag semantics. The basic idea of this procedure, called QCC (Query Containment Checker) is to use only a finite set of databases built from the body of the query Q to be tested for containment in some other query. This set of databases is called the *Canonical Database Set of* Q [3], in short: $CDBS(Q)$. For queries Q_1 and Q_2, either equality or inequality queries, under set or bag semantics, we show the following equivalence for $Q_1 \leq Q_2$ (i.e., the collection of answers to $Q_1$is contained in the collection of answers to Q_2, for all possible extensional databases):

$$Q_1 \leq Q_2 \iff \forall d \in CDBS(Q_1), Q_1(d) \subseteq Q_2(d)$$

Thus, containment can be effectively checked by using a finite, typically a small number of dedicated databases with relatively small extensions.

The rest of this paper is organized as follows. Some preliminary definitions are given in Section 2. Section 3 describes the basic procedure for testing containment and section 4 outlines its use for four orthogonal types of containment. We conclude with a short summary and an outlook.

2 Preliminary definitons

In the theory of relational algebra, a *conjunctive query* is a query using only Selection, Projection and Cartesian product operations. In Datalog terminology, a conjunctive query is a safe, nonrecursive rule whose predicates in the body are defined over EDB (Extensional Database) predicates [13]. Depending on the presence or absence of built-in predicates other than $=$ in the bodies, we distinguish two types of conjunctive queries:

Equality queries where built-in predicates, except equality, are not allowed. The general form of an equality query is $q(\vec{X})$:- $p_1(\vec{Y_1}), \ldots, p_n(\vec{Y_n})$. where $q(\vec{X})$ is the *query predicate* and every $p_i(\vec{Y_i})$ is an *ordinary predicate* defined over EDB predicates.

Inequality queries where also built-in predicates other than equality may occur in the body. An inequality query, in its general form, is a Datalog rule of the form

$$q(\vec{X}) \text{ :- } p_1(\vec{Y_1}), \ldots, p_n(\vec{Y_n}), F_1, \ldots, F_k.$$

where q, $\vec{X}$, p_i's, and $\vec{Y_i}$'s are defined as above, and every F_j is a built-in predicate of the form $X\theta Y$, being X and Y either variables that appear in an ordinary predicate, or constants of the domain (but not both constants), and $\theta \in \{=, \neq, <, \leq, >, \geq\}$.

Queries written as Datalog rules are based on the well-known concept of assignment mappings, denoting the derivation of facts as answers. An *assignment mapping* [13] τ from a query Q to a database D is a function from the symbols of Q to those of D; τ is the identity in the predicate names and constants, and it must map every ordinary predicate in the body of Q to a fact in D. If the query has built-in predicates, the application of the assignment mapping to them must produce a formula that evaluates to *true*. The derived fact corresponds to the application of the mapping to the head of the rule. Let $Q(D)$ represent the collection of answers to Q in D, i.e. the facts obtained by applying Q to D. Under set or bag semantics, $Q(D)$ is a set, or bag, respectively.

The application of a query is similar for equality and inequality queries, but, as we shall see, it is different under set and bag semantics. Under set semantics, both the relations (tables) in a database and the result of answering a query are sets of facts or tuples. A tuple or ground fact in an EDB (Extensional Database) is represented, under set semantics, as a predicate of the form $p(A_1, \ldots, A_l)$. Under bag semantics, a relation in a database is a *bag* or multiset of facts, i.e., every fact has a certain number of occurrences. In other words, each fact has an associated integer that indicates its *multiplicity*. So, any relation can be represented as a set of elements of the form $p(A_1, \ldots, A_l; [m])$, where $p(A_1, \ldots, A_l)$ is the fact and m is its *multiplicity*. The multiplicity of a fact in a relation p of a database D is represented as $|p(A_1, \ldots, A_l)|_D$. Similarly, for a query Q, we represent the multiplicity of an answer fact t in $Q(D)$ by $|t|_{Q(D)}$. If a fact does not occur in p or $Q(D)$, then its respective multiplicity is 0.

Let Q_1 be a query of the form $q(\vec{X})$:- $p_1(\vec{Y_1}), \ldots, p_n(\vec{Y_n})$, and D be a database. Assume there are l assignment mappings $\tau_1, \ldots, \tau_l$ from Q to D that return the derived answer fact t. That is,

$$t = \tau_1(q(\vec{X})) = \cdots = \tau_l(q(\vec{X})).$$

Every $p_i(\vec{Y_i})$ is mapped by any τ_j to a fact $p_i(\vec{A_i})$ in the database D. Let us abbreviate the multiplicity of $p_i(\vec{A_i})$ as m_{ji}:

$$m_{ji} = |\tau_j(p_i(\vec{Y_i}))|_D = |p_i(\vec{A_i})|_D$$

The multiplicity of t using only the mapping τ_j is computed by multiplying the multiplicities m_{ji}'s of the facts reached by the atoms in Q_1 using the mapping τ_j:

$$m_j = \prod_{i=1}^{n} |\tau_j(p_i(\vec{Y_i}))|_D$$

The final multiplicity of t is computed by adding the multiplicities of t obtained by every individual mapping τ_j:

$$|t|_{Q_1(D)} = \sum_{j=1}^{l} m_j = \sum_{j=1}^{l} \left(\prod_{i=1}^{n} |\tau_j(p_i(\vec{Y_i}))|_D \right)$$

A conjunctive query Q_1 is set-contained in a conjunctive query Q_2 (represented $Q_1 \leq_s Q_2$) if and only if, for all databases D, $Q_1(D) \subseteq_s Q_2(D)$, i.e., the set of facts returned by Q_1 is a subset of the set of facts returned by Q_2:

$$Q_1 \leq_s Q_2 \Longleftrightarrow \forall D,\ Q_1(D) \subseteq_s Q_2(D)$$

.

Two conjunctive queries Q_1 and Q_2 are set-equivalent, $Q_1 \equiv_s Q_2$, iff $Q_1 \leq_s Q_2$ and $Q_2 \leq_s Q_1$.

$$Q_1 \equiv_s Q_2 \Longleftrightarrow Q_1 \leq_s Q_2 \wedge Q_2 \leq_s Q_1$$

Note that we use the symbol $\subseteq_s$ to represent the subset relationship, instead of the usual $\subseteq$, in order to distinguish it from the subbag ($\subseteq_b$) relationship, defined below. It is fundamental to test query containment under bag semantics.

A bag B is a *subbag* of another bag B' if and only if every element t in B is also in B', with at least the same multiplicity as in B: $B \subseteq_b B' \iff \forall t \in B,\ |t|_B \leq |t|_{B'}$.

The containment of conjunctive queries under bag semantics (bag containment or b-containment, for short) is defined as follows. A query Q_1 is b-contained in a query Q_2, represented $Q_1 \leq_b Q_2$, if and only if, for all databases D, $Q_1(D) \subseteq_b Q_2(D)$. That is, the result obtained by applying Q_1 to any database is a subbag of the result obtained by Q_2 applied to the same database:

$$Q_1 \leq_b Q_2 \iff \forall D, Q_1(D) \subseteq_b Q_2(D)$$

$$Q_1 \leq_b Q_2 \iff \forall t, D\ |t|_{Q_1(D)} \leq |t|_{Q_2(D)}.$$

Query equivalence under bag semantics is defined naturally as mutual bag-containment: Two conjunctive queries Q_1 and Q_2 are bag-equivalent (represented $Q_1 \equiv_b Q_2$) iff $Q_1 \leq_b Q_2$ and $Q_2 \leq_b Q_1$.

$$Q_1 \equiv_b Q_2 \iff Q_1 \leq_b Q_2 \land Q_2 \leq_b Q_1$$

3 General description of the procedure

The *Query Containment Checker* (QCC) is a general procedure for deciding whether a query Q_1 is contained in another query Q_2.

Basically, QCC consists of the following 3 steps:

1. Build $CDBS(Q_1)$, the canonical database set for the query Q_1.
2. Apply Q_1 and Q_2 to all canonical databases $d \in CDBS(Q_1)$.
3. Test if $\forall d \in CDBS(Q_1),\ Q_1(d) \subseteq Q_2(d)$.

These three steps are essentially the same for the four cases enumerated above (under set or bag semantics, for equality or inequality queries), but there will be some particularization for each specific case.

According to the first step, we begin with describing the canonical database set for a query Q_1. The idea behind canonical databases [3] is to finitely capture all assignment mappings that could possibly be applied from a query Q to any database. The *Canonical Database Set* for a query Q ($CDBS(Q)$) is a finite set of databases with uninterpreted constant symbols in its facts. It captures all patterns of equalities and inequalities among all possible constants to which the variables of Q are mapped when Q is applied to the database.

The algorithm for building $CDBS(Q)$ can be divided into two steps. The first, which is common for our four cases of conjunctive query containment, builds the canonical databases. These are particularized in the second step, so that they are suitable for testing a given specific case. That is done by adding some constraints to the canonical databases, for the case of inequality queries, and/or assigning a symbolic multiplicity to the database facts, for the case of bag semantics. The case of equality queries and set semantics, being the normal one, needs no further adaptation.

3.1 Definitions

Let Q be a conjunctive inequality query of the form

$$Q:\ q(\vec{X}) \text{ :- } p_1(\vec{Y_1}), \ldots, p_l(\vec{Y_l}), K_1, \ldots, K_n.$$

¿From now on, we use this general query definition throughout, bearing in mind that, by definition, an equality query is just a special case of an inequality query. Furthermore, we define:

- $db(Q)$: is the set of ordinary (EDB) predicates in the body of Q.
- V_Q: is an (arbitrary, but fixed) ordering $\langle V_1, \cdots, V_q \rangle$ of the set of all variables appearing in the predicates in $db(Q)$.
- A_Q: is a set of q fresh, uninterpreted constants, q being the cardinality of V_Q.
- *Q-mapping:* A Q-mapping θ_i from V_Q to A_Q is a q-tuple $\theta_i = (A_{i_1}, ..., A_{i_q})$, where $A_{ij} \in A_Q$ and $1 \leq i_1, i_2, ..., i_q \leq q$. It represents the mapping $\theta_i(V_1) = A_{i_1}, ..., \theta_i(V_q) = A_{i_q}$.

  A Q-mapping can be applied to a predicate in the body of Q: let θ be a Q-mapping, and $p_i(Y_1, ..., Y_{r_i})$ be a predicate in Q. The application of θ to $p_i(Y_1, ..., Y_{r_i})$ is defined as the fact $p_i(\theta(Y_1), ..., \theta(Y_{r_i}))$.
- Canonical database $d_i = \theta_i(db(Q))$: It is the application of the Q-mapping θ_i to the set of ordinary predicates in the body of Q, that is,

  $d_i = \theta_i(db(Q)) = \{p_k(\theta_i(Y_1), ..., \theta_i(Y_{r_k})) \mid p_k(Y_1, ..., Y_{r_k}) \in db(Q)\}$

  d_i is a database with uninterpreted constants, which represents a pattern of equalities and inequalities among the variables of the query that are mapped to these constants. All uninterpreted constants must be mutually distinct ($A_j \neq A_k, \forall j, k\ 1 \leq j \neq k \leq q$)).
- *Canonical fact t_{d_i}:* Let Q be the query $q(\vec{X}) \text{ :- } p_1(\vec{Y_1}), \ldots, p_l(\vec{Y_l}), K_1, \ldots, K_n$, and d_i be the canonical database obtained using the Q-mapping θ_i. Then, the canonical fact t_{d_i} is the fact $\theta_i(q(\vec{X}))$.

3.2 Building the canonical database set for a query

The full-fledged algorithm to build $CDBS(Q_1)$ has been elaborated in [2, 11]. Its main idea is to determine all different Q-mappings (modulo isomorphisms) applicable to the body of Q_1 and apply them to $db(Q_1)$. However, instead of describing and using the formal algorithm, let us look at an example to convey the intuition of how to build $CDBS(Q_1)$.

Example 3.1 Let the query Q_1 be $q(X,Y,Z) :- r(X,U),\ r(U,Z),\ p(U,Y)$. Letters A,B,C,D represent uninterpreted constants, i.e., they identify values to which the variables in Q_1 could be mapped. That is, $A_Q = \langle A,B,C,D \rangle$.

There are 4 variables in the body of Q_1, which will be mapped to 4 uninterpreted constants. Therefore, there are $4^4 = 256$ possible mappings. For example, all the variables can be mapped to A, all of them mapped to B, three of them mapped to A and one to B, etc. However, employing all possible mappings is redundant. For example, all the variables mapped to A or mapped to B represent the same pattern of equalities, thus only one of them is needed. The following cases list all the different patterns of equalities among the variables in the body of Q_1 when they are mapped to 4 uninterpreted constants.

Case 1: Each variable in Q_1 is mapped to a different value. Then the canonical database $\theta_1(db(Q_1))$ shown on Table 1 is generated.

Table 1: Canonical databases generated for case 1

NAME	$\theta_i(db(Q_1))$		MAPPING	t_{d_i}
	r	p	X Y Z U	q
θ_1	(A,D) (D,C)	(D,B)	A B C D	ABC

Case 2: Three variables are mapped to the same value, the other one is mapped to a different value. Table 2 shows the four corresponding canonical databases.

Table 2: Canonical databases generated for case 2

NAME	$\theta_i(db(Q_1))$		MAPPING	t_{d_i}
	r	p	X Y Z U	q
θ_2	(A,B) (B,A)	(B,A)	A A A B	AAA
θ_3	(A,A) (A,B)	(A,A)	A A B A	AAB
θ_4	(A,A)	(A,B)	A B A A	ABA
θ_5	(B,A) (A,A)	(A,A)	B A A A	BAA

Case 3: Two variables are mapped to the same value and the other two are mapped to another value. Thus, 3 canonical databases are generated, as shown in Table 3.

Table 3: Canonical databases generated for case 3

NAME	$\theta_i(db(Q_1))$		MAPPING	t_d
	r	p	X Y Z U	q
θ_6	(A,B) (B,B)	(B,A)	A A B B	AAB
θ_7	(A,B) (B,A)	(B,B)	A B A B	ABA
θ_8	(A,A) (A,B)	(A,B)	A B B A	ABB

Case 4: Two variables are mapped to the same value and the other two are mapped to different values, producing the 6 canonical databases shown in Table 4.

Table 4: Canonical databases generated for case 4

NAME	$\theta_i(db(Q_1))$		MAPPING	t_{d_i}
	r	p	X Y Z U	q
θ_9	(A,C) (C,B)	(C,A)	A A B C	AAB
θ_{10}	(A,C) (C,A)	(C,B)	A B A C	ABA
θ_{11}	(A,A) (A,C)	(A,B)	A B C A	ABC
θ_{12}	(B,C) (C,A)	(C,A)	B A A C	BAA
θ_{13}	(B,A) (A,C)	(A,A)	B A C A	BAC
θ_{14}	(B,A) (A,A)	(A,C)	B C A A	BCA

Case 5: The four variables are mapped to the same value. So, only one canonical database, shown in Table 5, is generated in this case.

Table 5: Canonical databases generated for case 5

NAME	$\theta_i(db(Q_1))$		MAPPING	t_d
	r	p	X Y Z U	q
θ_{15}	(A,A)	(A,A)	A A A A	AAA

To summarize, we have generated 15 canonical databases for Q_1, using all possible Q-mappings that are mutually non-isomorphic. With four variables, there are 256 different possible canonical databases, but there are only 15 really different (non-isomorphic) ones. In [2], a procedure to compute the number of databases (the number of non isomorphic Q-mappings) in terms of the number of variables in the conjunctive query is described. □

4 Use of QCC to test different types of containment

The use of canonical databases in QCC, the general procedure to test whether a conjunctive query Q_1 is contained in a conjunctive query Q_2 is based on the fundamental property that the set (or bag) of facts of a database D reached by an assignment mapping from Q_1 to D (used to derive a new fact) is always isomorphic to some canonical database d_i in $CDBS(Q_1)$. QCC consists of the following three steps.

Step 1: Build $CDBS(Q_1)$, the set of canonical databases for the query Q_1.

The exact algorithm to build $CDBS(Q_1)$ is described in [11]. Its last step is the adaptation of the canonical databases to check each specific case of query containment. For example, $CDBS(Q_1)$ must include multiplicities in the facts to check bag containment of equality queries, or constraints to check set and bag containment of inequality queries.

Step 2: Apply Q_1 and Q_2 to all canonical databases $d \in CDBS(Q_1)$ in order to derive the canonical fact t_d.

The application of both queries intend to derive only the canonical fact t_d. Besides, all the mappings from either Q_1 or Q_2 that derive t_d will be considered. However, under set semantics, we only need to apply Q_2 to the databases since, as we will see, Q_1 always obtains the canonical fact, by construction of $CDBS(Q_1)$.

Step 3: Check the containment.

Query containment holds (i.e., $Q_1 \leq Q_2$) if and only if Q_2 obtains the canonical fact t_d from all d in $CDBS(Q_1)$. Under bag semantics, Q_2 must obtain it with at least the same multiplicity as Q_1.

The main advantage of QCC is that these three steps are invariant for many different kinds of containment, such as equality and inequality queries under set or bag semantics. There will be, of course, some adaptations for each specific case, but conceptually, every step of the procedure does the same in all cases. Therefore, QCC is a *general* procedure for testing conjunctive query containment. Let us see now how this procedure is applied to test each type of continment.

4.1 Use of QCC to test set containment of equality queries

Although this problem has already been solved by Chandra and Merlin [6], we show that QCC also works for this case.

Let Q_1 and Q_2 be two equality queries under set semantics. The procedure to test if $Q_1 \leq_s Q_2$ is the following.

Step 1: Build $CDBS(Q_1)$.

For this particular case, canonical databases do not need any further transformation.

Step 2: Apply Q_1 and Q_2 to every $d \in CDBS(Q_1)$ in order to obtain the canonical fact.

There is no real need to apply Q_1 to each canonical database because, by the way they were built, we already know that Q_1 obtains the canonical fact, using an assignment mapping isomorphic to the respective Q-mapping. So, we only try to find an assignment mapping from Q_2 to every canonical database to derive the canonical fact.

Step 3: Testing containment. If Q_2 obtains the canonical fact from every canonical database $d \in CDBS(Q_1)$, then $Q_1 \leq_s Q_2$, else Q_1 is not contained in Q_2.

The use of QCC for this case is very similar to the solution by Chandra and Merlin [6]. Note that the canonical database d_i built using a Q-mapping that maps every variable of Q_1 to a different uninterpreted constant is isomorphic to the body of Q_1. Thus, finding an assignment mapping from Q_2 to such d_i (that is, applying Q_2 to d_i) is exactly the same problem as finding a containment mapping from Q_2 to Q_1, which is the only needed condition for testing set containment of equality queries shown in [6].

4.2 Use of QCC to test set containment of inequality queries

In this chapter, the inequality queries Q_1 and Q_2 will be represented as

$$Q_1 : q(\vec{W}) \text{ :- } p_1(\vec{Y_1}), \ldots, p_l(\vec{Y_l}), K_1, \ldots, K_n.$$

$$Q_2 : q(\vec{V}) \text{ :- } p_1(\vec{Z_1}), \ldots, p_k(\vec{Z_k}), F_1, \ldots, F_m.$$

where the p_i's are ordinary predicates, and K_i's and F_i's are built-in predicates.

The three steps of QCC to check set containment of inequality queries are the following.

4.2.1 Step 1: Build $CDBS(Q_1)$

Let Q_1 be an inequality query, and D a ground database. In order for Q_1 to obtain a fact from D, there must exist (at least) one assignment mapping τ from Q_1 to D such that the constants in the tuples of D reached by the variables in the body of Q_1 using τ satisfy the constraints expressed by the built-in predicates of Q_1.

Given that canonical databases are used to test query containment, we are interested in those canonical databases from which Q_1 derives the canonical fact (if Q_1 does not obtain it, the fact that Q_2 derives it or not is irrelevant for the containment). Therefore, the canonical databases used to test set containment of inequality queries will have associated some constraints that ensure that Q_1 will obtain the canonical fact from them. These constraints will be denoted $constraints(d)$ for any canonical database set, and are composed of two sets of constraints:

1. All uninterpreted constants must represent different values for constants. Therefore, $\forall i, j\ (1 \leq i \neq j \leq q)\ \ A_i \neq A_j$. ($q$ is the cardinality of V_Q, i.e., the number of variables in Q_1).
2. The second set of constraints is built by applying the Q-mapping θ_i used to build the canonical database $d_i = \theta_i(db(Q_1))$ to the built-in predicates, $(\theta_i(K_1) \wedge \cdots \wedge \theta_i(K_n))$. This set of constraints reflects the built-in predicates in the body of Q_1.

If a canonical database d_i does not satisfy $constraints(d_i)$, it will not be used to testing set containment, because it is not possible to build a ground database isomorphic to it from which Q_1 obtains the canonical fact.

Example 4.1 Let Q_1 be the following query:

$$Q_1\ :\ q(X,Y)\ \text{:-}\ \ r(X,Y), p(U,V), p(V,U), X > Y.$$

One of the canonical databases built from Q_1 (using a Q-mapping that maps every variable to a different uninterpreted constant) is the following.

CDB		t_{d_i}	Q-mapping
r	p	q	X Y U V
A B	C D D C	A B	A B C D

The following formula is associated to this canonical database

$constraints(d) = (A \neq B) \wedge (A \neq C) \wedge (A \neq D)$
$\wedge (B \neq C) \wedge (B \neq D) \wedge (C \neq D) \wedge (A > B)$.

This formula is satisfiable, so it will be considered. If $constraints(d)$ were unsatisfiable, this database would not be taken into account for containment testing. □

4.2.2 Step 2: Apply Q_1 and Q_2 to all canonical databases

This step applies Q_1 and Q_2 to all canonical databases trying to derive the canonical fact t_d. By adding $constraints(d)$ to every canonical database d, we ensure that Q_1 always obtains t_d, therefore there is no need to apply Q_1 to each $d \in CDBS(Q_1)$. However, it is necessary to apply Q_2 to all canonical databases in order to derive the canonical fact.

Intuitively, the method used to test if $t_d \in Q_2(d)$ is the following:

1. Find $\tau_1, \ldots, \tau_l$, the assignment mappings from Q_2 to d.
2. Build a formula F: Start with $F = constraints(d)$, and add the negation of the application of the assignment mappings $\tau_1, \ldots, \tau_l$ to the built-in predicates of Q_2:
$$F = constraints(d)\ \wedge\ \neg[\tau_1(F_1) \wedge \ldots \wedge \tau_1(F_m)]\ \wedge\ \ldots\ \wedge\ \neg[\tau_l(F_1) \wedge \ldots \wedge \tau_l(F_m)].$$
3. If F is unsatisfiable, then Q_2 always derives the canonical fact. Otherwise, Q_2 *not always* derives the canonical fact (which is enough for the containment not to hold).

The following example illustrates the test of membership of a tuple in $Q_2(D)$.

Example 4.2 Consider the query Q_1 and canonical database from Example 4.1, and the following query Q_2:

$$Q_2 : q(X,Y)\ \text{:-}\ \ r(X,Y), p(U,V), U \leq V.$$

There are two ways to map the ordinary subgoals of Q_2 to d in a way to obtain the canonical fact $q(A, B)$. These are

$$\tau_1(X) = A;\ \ \tau_1(Y) = B;\ \ \tau_1(U) = C;\ \ \tau_1(V) = D$$

$$\tau_2(X) = A;\ \ \tau_2(Y) = B;\ \ \tau_2(U) = D;\ \ \tau_2(V) = C$$

We use both mappings to test whether $q(A, B)$ belongs to $Q_2(d)$, checking if the following formula is not satisfiable:

$$constraints(d) \wedge \neg(\tau_1(U \leq V)) \wedge \neg(\tau_2(U \leq V))$$

The procedure described in [4] can be used to check the satisfiability of this kind of formulas. However, due to the simplicity of the formula, this check can be done directly, in this example, . The above formula is not satisfiable, since

$$constraints(d) \wedge \neg(\tau_1(U \leq V)) \wedge \neg(\tau_2(U \leq V))$$
$$\equiv$$
$$constraints(d) \wedge \neg(C \leq D) \wedge \neg(D \leq C)$$
$$\equiv$$
$$constraints(d) \wedge (C > D) \wedge (D > C)$$
$$\equiv$$
$$unsatisfiable$$

The unsatisfiability of the formula means that, for any ground substitution α, Q_2 always obtains $\alpha(t_{d_i})$ from $\alpha(d_i)$, because either $C \leq D$ or $D \leq C$ is true. That makes possible to apply at least one of the assignment mappings from Q_2 to d, satisfying the built-in predicates in Q_2. Therefore, $q(A, B) \in Q_2(d)$. □

4.2.3 Step 3: Testing set containment

In the previous step, we showed how to check if the canonical fact t_d belongs to $Q_2(d)$ for any canonical database $d \in CDBS(Q_1)$.

If, for all canonical databases $d \in CDBS(Q_1)$, $t_d \in Q_2(d)$, then $Q_1 \leq_s Q_2$, otherwise the containment does not hold:

$$Q_1 \leq_s Q_2 \Longleftrightarrow \forall d \in CDBS(Q_1),\ t_d \in Q_2(d)$$

4.3 Use of QCC to test bag containment of equality queries

Using QCC to test bag containment of equality queries, we only need to test containment over the set of canonical databases built from the body of Q_1. Therefore, for a suitable bag containment test, every canonical database must have a symbolic multiplicity associated to each of its facts.

The complete description of the procedure to test bag containment of equality queries has been given in [2]. We shall show here how this procedure perfectly fits into the 3 steps of QCC.

4.3.1 Step 1: Build $CDBS(Q_1)$

In order to test bag containment of equality queries, canonical databases will be adapted to include symbolic multiplicities in their facts. Then, for each canonical database $d_i = \theta_i(db(Q_1))$, we add a symbolic multiplicity to each fact in it. Every fact will be of the form

$$d_i = \{p(\theta_i(Y_1), \ldots, \theta_i(Y_l); [m]) \mid p(Y_1, \ldots, Y_l) \in db(Q_1)\}$$

where each m is a new, different identifier that represents the multiplicity of the fact $p(\theta_i(Y_1), \ldots, \theta_i(Y_l))$ in d_i.

Example 4.3 Let Q_1 and Q_2 be the following equality queries.

$$Q_1: \ q(X) \coloneq p(X), r(Y,Z), r(Z,Y).$$

$$Q_2: \ q(X) \coloneq p(X), r(U,V), r(U,V).$$

The set of canonical databases for Q_1 is shown in Table 6.

□

Table 6: Canonical Database set for Q_1

	Q-mappings			CDB		t_d
	X	Y	Z	p	r	q
d_1	A	A	A	$A[m_p]$	$AA[m_r]$	A
d_2	A	A	B	$A[m_p]$	$AB[m_{r1}]$ $BA[m_{r2}]$	A
d_3	A	B	A	$A[m_p]$	$BA[m_{r1}]$ $AB[m_{r2}]$	A
d_4	B	A	A	$B[m_p]$	$AA[m_r]$	B
d_5	A	B	C	$A[m_p]$	$BC[m_{r1}]$ $CB[m_{r2}]$	A

4.3.2 Step 2: Apply Q_1 and Q_2 to all canonical databases

In this step, Q_1 and Q_2 will be applied to every $d_i \in CDBS(Q_1)$ in order to obtain the canonical fact t_{d_i} with a certain multiplicity. Note that there can be more than one assignment mapping from either Q_1 or Q_2 that derive the canonical fact. In that case, all assignment mappings must be taken into account to compute the final multiplicity of the canonical fact.

Example 4.4 (Continued from Example 4.3).

The column CDB in Tables 7 and 8 shows the canonical databases, whose facts have a symbolic multiplicity. The column t_d shows the canonical fact for each database. The last column shows how Q_1 (in Table 7) and Q_2 (in Table 8) are applied to each canonical database to derive the canonical fact. The multiplicities of the canonical facts obtained by Q_1 and Q_2 are computed as shown in Section 2. Let us show how to compute the multiplicity of $t_{d_2} = q(A)$ obtained by Q_1:

There are two assignment mappings τ_1 and τ_2 from Q_1 to d_2:

$$\tau_1(X) = A; \quad \tau_1(Y) = A; \quad \tau_1(Z) = B$$

$$\tau_2(X) = A; \quad \tau_2(Y) = B; \quad \tau_2(Z) = A$$

Applying τ_1 and τ_2 to the body of Q_1, we get

$$\tau_1(p(X)) = p(A;[m_p]); \quad \tau_1(r(Y,Z)) = r(A,B;[m_{r1}]); \ \tau_1(r(Z,Y)) = r(B,A;[m_{r2}])$$

$$\tau_2(p(X)) = p(A;[m_p]); \quad \tau_2(r(Y,Z)) = r(B,A;[m_{r2}]); \ \tau_2(r(Z,Y)) = r(A,B;[m_{r1}])$$

Therefore, the multiplicity of $p(A)$ using τ_1 is $m_p m_{r1} m_{r2}$; using τ_2 is $m_p m_{r2} m_{r1}$. Then, the final multiplicity is

$$|t_{d_2}|_{Q_1(d_2)} = |q(A)|_{Q_1(d_2)} = m_p m_{r1} m_{r2} + m_p m_{r2} m_{r1}.$$

The rest of the multiplicities is calculated in the same way.

□

4.3.3 Step 3: Testing bag containment

Up to this point, we have created $CDBS(Q_1)$ and computed the multiplicity of every canonical fact obtained by Q_1 and Q_2.

This last step of QCC compares the polynomials that represent the multiplicities with which Q_1 and Q_2 obtain every canonical fact. If Q_2 obtains the canonical facts with at least the same multiplicity as Q_1 for all canonical databases, then the containment holds, else $Q_1 \not\leq_b Q_2$. Thus, the containment problem is reduced to a polynomial comparison. For this comparison, the

Table 7: Multiplicities of the canonical facts obtained by Q_1.

	CDB		t_d	Applying Q_1	
	p	r	q	As. mapps.	$\|t_d\|_{Q_1}$
				X Y Z	
d_1	$A[m_p]$	$AA[m_r]$	A	A A A	$m_p m_r^2$
d_2	$A[m_p]$	$AB[m_{r1}]$ $BA[m_{r2}]$	A	A A B A B A	$m_p m_{r1} m_{r2} +$ $m_p m_{r2} m_{r1}$
d_3	$A[m_p]$	$BA[m_{r1}]$ $AB[m_{r2}]$	A	A B A A A B	$m_p m_{r1} m_{r2} +$ $m_p m_{r2} m_{r1}$
d_4	$B[m_p]$	$AA[m_r]$	B	B A A	$m_p m_r^2$
d_5	$A[m_p]$	$BC[m_{r1}]$ $CB[m_{r2}]$	A	A B C A C B	$m_p m_{r1} m_{r2} +$ $m_p m_{r2} m_{r1}$

Table 8: Multiplicities of the canonical facts obtained by Q_2.

	CDB		t_d	Applying Q_2	
	p	r	q	As. mapps.	$\|t_d\|_{Q_2}$
				X U V	
d_1	$A[m_p]$	$AA[m_r]$	A	A A A	$m_p m_r^2$
d_2	$A[m_p]$	$AB[m_{r1}]$ $BA[m_{r2}]$	A	A A B A B A	$m_p m_{r1}^2 +$ $m_p m_{r2}^2$
d_3	$A[m_p]$	$BA[m_{r1}]$ $AB[m_{r2}]$	A	A B A A A B	$m_p m_{r1}^2 +$ $m_p m_{r2}^2$
d_4	$B[m_p]$	$AA[m_r]$	B	A A A	$m_p m_r^2$
d_5	$A[m_p]$	$BC[m_{r1}]$ $CB[m_{r2}]$	A	A B C A C B	$m_p m_{r1}^2 +$ $m_p m_{r2}^2$

method presented in [3] can be used. In Example 4.4, for all canonical databases d, $|t_d|_{Q_1(d)} \leq |t_d|_{Q_2(d)}$. Thus, for this example, $Q_1 \leq_b Q_2$.

4.4 Use of QCC to test bag containment of inequality queries

The first step of QCC for this type of containment must adapt the canonical databases including multiplicities in their facts and constraints that specify which assignment mappings from Q_1 to any $d_i \in CDBS(Q_1)$ can be applied. The second step applies Q_1 and Q_2 to all canonical databases, obtaining the canonical facts with some multiplicities. The third step tests bag containment by comparing the polynomials that represent those multiplicities.

4.4.1 Step 1: Build $CDBS(Q_1)$

In order to test bag containment, canonical databases will include multiplicities in their facts, as in the previous chapter. Once the canonical databases are build, Q_1 and Q_2 are applied to each one of them, in order to derive the canonical fact. The main goal, as in the previous section, is to obtain a polynomial that represents this multiplicity. In order to obtain it, all assignment mappings must be applied, because each one contributes a monomial to the polynomial representing the multiplicity. However, due to the presence of inequalities in the queries, it is not clear when each assignment mapping can be applied. Adding $constraints(d)$ to each canonical database, as for testing set containment, is not restrictive enough to let us know which assignment mappings from Q_1 to each canonical database can be applied, as Example 4.5 shows.

Example 4.5 Consider the following query Q_1 and the canonical database d_1:

$$Q_1 : \ q(X,Y) \text{ :- } \ r(X,Y), p(X,Z), p(Y,V), X < V.$$

	Q-Mapping	$\theta_1(db(Q_1))$		t_d
	X Y Z V	r	p	q
d_1	A A B C	AA$[m_{r1}]$	AB$[m_{p1}]$ AC$[m_{p2}]$	AA

$$constraints(d_1) = (A \neq B \wedge B \neq C \wedge A \neq C) \wedge A < C$$

There are 4 possible ways of applying Q_1 over d_1, using the assignment mappings τ_1 through τ_4, shown in the following table:

	$r(X,Y)$	$p(X,Z)$	$p(Y,V)$	$X < V$
τ_1	$r(A,A)$	$p(A,B)$	$p(A,C)$	$A < C$
τ_2	$r(A,A)$	$p(A,B)$	$p(A,B)$	$A < B$
τ_3	$r(A,A)$	$p(A,C)$	$p(A,B)$	$A < B$
τ_4	$r(A,A)$	$p(A,C)$	$p(A,C)$	$A < C$

The mapping τ_1 is isomorphic to θ_1 and it can always be applied, because the application of τ_1 to the built-in predicates of Q_1, $\tau_1(X < V) = A < C$, is already in $constraints(d_1)$. The assignment mapping τ_4 applied to the built-in predicate of Q_1 is also $A < C$, thus τ_4 can always be applied.

However, the application of τ_2 and τ_3 to $X < V$ is $A < B$, which is not in $constraints(d_1)$, so it is not possible to decide if τ_2 and τ_3 can be applied to d_1. Therefore, the multiplicity of the canonical fact obtained by Q_1 is either $m_{r1}m_{p1}m_{p2} + m_{r1}m_{p2}^2$m when only τ_1 and τ_4 can be applied, or $m_{r1}m_{p1}m_{p2} + m_{r1}m_{p1}^2 + m_{r1}m_{p2}m_{p1} + m_{r1}m_{p2}^2$ when the 4 mappings can be applied. Therefore, the multiplicity of the canonical fact obtained by Q_1 is not exactly known. □

Since the multiplicity of the canonical fact obtained by Q_1 will be compared with the multiplicity obtained by Q_2, it is clear that $constraints(d)$, defined identically as for testing set containment of inequality queries, are not restrictive enough to specify when a given set of assignment mappings from Q_1 to each d_i can be applied.

The idea to solve this problem is to adapt the canonical databases so that Q_1 obtains the canonical fact with a predefined multiplicity (that is, using a fixed and predetermined set of assignment mappings). This can be done by splitting each canonical database into several ones, with exactly the same facts, but adding the constraints necessary to fix the assignment mappings that can be applied from Q_1.

Example 4.6 Splitting d_1 from Example 4.5 into two canonical databases d_1^1 and d_1^2 that have the same bag of facts as d_1, but different sets of constraints, as shown below, achieves this goal.

$$\mathit{constraints}(d_1^1) = (A \neq B \wedge B \neq C \wedge A \neq C) \wedge A < C \wedge \neg(A < B)$$
$$\mathit{constraints}(d_1^2) = (A \neq B \wedge B \neq C \wedge A \neq C) \wedge A < C \wedge A < B$$

It is clear that the assignment mappings applicable to each canonical database (τ_1 and τ_4, but not τ_2 and τ_3, to d_1^1, and the 4 mappings to d_1^2) are completely specified with the new constraints. □

The assignment mappings from Q_1 to a canonical database are divided into two groups: one assignment mapping that can always be applied (the one isomorphic to the Q-mapping used to build the canonical database) and the remaining ones, denoted *possible mappings* and represented $\mathcal{M}$. Using $\mathcal{M}$, we can describe completely the constraints associated to canonical databases to test bag containment of inequality queries. The formula $\mathit{constraints}(d_i)$ is composed of three sets of constraints.

- Constraints that specify that all uninterpreted constants are different.
- Constraints that reflect the built-in predicates of Q_1.
- Constraints that specify which sets of mappings, from Q_1 to d_i, can be applied and which ones cannot.

  This set of constraints establishes which set of mappings $M_j \subset \mathcal{M}$ is applicable, by adding either $[\tau_i(K_1) \wedge \cdots \wedge \tau_i(K_n)]$, if the assignment mapping τ_i can be applied ($\tau_i \in M_j$), or $\neg[\tau_i(K_1) \wedge \cdots \wedge \tau_i(K_n)]$ if it cannot ($\tau_i \notin M_j$), $\forall i, 2 \leq i \leq l$, where K's are the built-in predicates of Q_1.

4.4.2 Step 2: Apply Q_1 and Q_2 to all canonical databases

This section shows how to apply Q_1 and Q_2 to a canonical database in order to derive *only* the canonical fact.

Application of Q_1 to a canonical database:

By construction of $CDBS(Q_1)$, the assignment mappings from Q_1 that are applicable to a canonical database d are known. Therefore, the multiplicity of the canonical fact t_d obtained by Q_1 is computed by adding the multiplicities obtained by each single mapping that is applicable from Q_1 to d.

Application of Q_2 to a canonical database:

Similiarly as for Q_1, there will be different sets of assignment mappings that can be applied from Q_2 to d such that Q_2 obtains the canonical fact. For each of these sets, a different multiplicity for the canonical fact is obtained, and all of them must be considered. In order to compute such multiplicities, it is necessary to study which sets of assignment mappings from Q_2 to d (to obtain the canonical fact) can be applied.

In this case, there is no guarantee that an assignment mapping isomorphic to the Q-mapping exists, thus all assingment mappings are *possible mappings*. Let $\mathcal{M}' = \{\tau_1, \ldots, \tau_m\}$ this set of possible mappings from Q_2 to d. All the subsets of $\mathcal{M}'$ are are parts of $\mathcal{M}'$:

$$P(\mathcal{M}') = \{\emptyset,\ \{\tau_1\}, \ldots,\ \{\tau_m\},\ \{\tau_1, \tau_2\}, \ldots,\ \{\tau_1, \ldots, \tau_m\}\} = \{M'_0, \ldots, M'_{2^m-1}\}.$$

It is necessary to test which sets of mappings can be applied from Q_2 to d. For each case, we shall build a formula that is the conjunction of $\mathit{constraints}(d)$ and the constraints that force a set of assignment mappings M'_j in $P(\mathcal{M}')$ to become applicable. These constraints are added to the formula in the following way:

- Let M'_j be a set of mappings in $P(\mathcal{M}')$.
- For each mapping $\tau_k \in \mathcal{M}'$, if $\tau_k \in M'_j$, then add $\tau_k(F_1 \wedge \ldots \wedge F_r)$ to the formula, otherwise add $\neg(\tau_k(F_1 \wedge \ldots \wedge F_r))$, where the F's are the built-in predicates of Q_2.
- The resulting formula must be satisfiable if the case needs to be considered. If the formula is unsatisfiable, it would mean that Q_2 cannnot obtain the canonical fact using exaclty the assignment mappings in M'_j, because the built-in predicates of Q_2 would not be satisfied. Therefore, if the formula is unsatisfiable, the case is discarded. For each case with a satisfiable formula, the multiplicity of the canonical fact obtained by Q_2 is the sum of the multiplicities obtained by Q_2 using only the mappings in M'_j. Let us denote this multiplicity as $(|t_d|_{Q_2(d)})_{M'_j}$.

We need to compute the multiplicity of the canonical fact for all possible cases because the multiplicity obtained by Q_1 must be compared with all of them in order to prove bag containment. A specially important case is the one that corresponds to the empty set of mappings. If it produces a satisfiable formula, it means that there is a possibility that none of the assignment mappings from Q_2 to d can be applied, so Q_2 does not derive the canonical fact (the multiplicity would be 0). In that case, the procedure can stop, concluding that bag containment does not hold.

4.4.3 Step 3: Testing bag containment

This step compares the multiplicity of the canonical fact obtained by Q_1 (which is unique for each canonical database) with all multiplicities (corresponding to the different sets of mappings) obtained by Q_2. If, for all cases (and for all databases), the multiplicity obtained by Q_2 is always at least as high as that obtained by Q_1, then the containment holds ($Q_1 \leq_b Q_2$), else $Q_1 \not\leq_b Q_2$.

5 Conclusion

Along the two independent dimensions of comparison operators in queries (first) and collection types of answers (second), we have picked the alternative cases of equality or inequalitiy queries, on one hand, and sets or bags of answers, on the other. The main contribution of the paper is the outline of an effective, uniform algorithm, called QCC, for testing query containment. It applies to each of the four different cases in the space spanned by the cases just mentioned. That is, QCC applies to each of the four different "perspectives" identified in the introduction. It builds on the concept of "canonical database set", as introduced in [3]. In particular, we have shown that QCC can be specialized to a procedure for testing containment of inequality queries under bag semantics, which, to our knowledge, hitherto has not been discussed in the literature. This case is more general and more interesting than the other three, since inequality queries are more general than equality queries, bags are more general than sets, and bags are default in commercial DBMS.

In future work, we are going to discuss an alternative method for checking set containment of inequality queries, using a theory of formula subsumption, and set it off against the proposal discussed in this paper. Also, we intend to investigate the applicability of the QCC approach to validating various database schema properties, as defined in [8], such as the "redundancy" of integrity constraints, which involves a generalization of the concept of query containment with regard to database integrity. Also, it should be interesting to target an extension of QCC to other collection types, e.g., the resource-oriented collections of Linear Logic [5], or the location- and communication-aware containers of the XML-based semi-structured data description language of RDF [1].

References

[1] http://www.w3.org/tr/pr-rdf-syntax/.

[2] Nieves R. Brisaboa. *Inclusión de Consultas Conjuntivas en la semántica de bolsas.* PhD thesis, Departamento de Computación, Facultade de Informática, Universidade da Coruña, A Coruña, Spain, May 1997.

[3] Nieves R. Brisaboa and Héctor J. Hernández. Testing bag-containment of conjunctive queries. *Acta Informatica*, 34:557–578, 1997.

[4] Nieves R. Brisaboa, Héctor J. Hernández, José R. Paramá, and Miguel R. Penabad. Containment of conjunctive queries with built-in predicates with variables and constants over any ordered domain. In *Advances in Databases and Information Systems (ADBIS'98)*, LNCS 1475, pages 46–57, Poland, 1998. Springer-Verlag.

[5] Iliano Cervesato, Joshua S. Hodas, and Frank Pfenning. Efficient resource management for linear logic proof search. In *Proc. Fifth Int'l Workshop Extensions of Logic Programming, ELP '96*, pages 67–81. Springer-Verlag LNAI 1050, 1996.

[6] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational databases. In *Proc. 9th ACM SIGACT Symp. on the Theory of Computing*, pages 77–90, New York, 1977.

[7] S. Chaudhuri and M. Y. Vardi. Optimization of real conjunctive queries. In *Proc. Twelfth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 59–70, Washington, DC, May 1993.

[8] Hendrik Decker, Ernest Teniente, and Toni Urp. How to tackle schema validation by view updating. In *Extending Database Technology (EDBT'98)*, LNCS 1057, pages 535–549. Springer-Verlag, 1996.

[9] Oscar H. Ibarra and Jianwen Su. On the containment and equivalence of database queries with linear constraints. In *PODS'97*, pages 32–43, Tucson, Arizona, 1997.

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

[11] Miguel R. Penabad. *A general procedure to test bag containment of conjunctive queries.* PhD thesis, Departamento de Computación, Facultade de Informática, Universidade da Coruña, A Coruña, Spain, May 2001.

[12] B. A. Trakhtenbrot. The imposibility of an algorithm for the decision problem for finite models. *Doklady Akademii Naurk SSR*, 70:569–572, 1950.

[13] Jeffrey D. Ullman. *Principles of Database and Knowledge-base Systems*, volume 1 and 2. Computer Science Press, 1988-1989.

[14] Ron van der Meyden. The complexity of querying indefinite data about linearly ordered domains. *Journal of Computer and System Sciences*, 54(1):113–135, 1997.