A general procedure to test conjunctive query containment

Miguel Rodríguez Penabad

Ph.D Thesis

Departamento de Computación
Universidade da Coruña

Directed by: Nieves Rodríguez Brisaboa

Ph.D. Thesis directed by
Tese doutoral dirixida por

Nieves Rodríguez Brisaboa
Departamento de Computación
Facultade de Informática
Universidade da Coruña
15071 A Coruña (España)
Tel: +34 981 167000 ext. 1243
Fax: +34 981 167160
brisaboa@udc.es

Agradecementos

O primeiro agradecemento vai para a miña familia. A meus pais, José e Lola, a miña irmá Elba ("mamá número dous") e a meu irmán Juan. A meu cuñado Tino e meus sobriños Oscar, Nuria e Lucía. Polo seu cariño e apoio pero, sobre todo, por aguantarme.

A Nieves, a miña directora de tese e directora do Laboratorio de Bases de Datos da Universidade da Coruña, por toda a súa axuda tanto dentro coma fóra da tese. Ós demáis membros, actuais ou pasados, do laboratorio: Jose, Eva, mon (en minúsculas, como ti queres), Ángeles, Mariajo, Charo, Miguel, José Ramón, Antonio (Fari) e Tony. Gracias a todos, e espero non esquecer a ninguén. Gracias tamén ó Doutor Héctor Hernández e á xente do Departamento de Computer Science da New Mexico State University por toda a súa axuda durante a miña estancia nesa universidade, onde empecei o desenrolo desta Tese.

E finalmente, anque non menos importante, a tódolos amigos e amigas que tamén estiveron ahí cando os necesitaba: Carlos, Ana, Manuel, Miryam, David, Luli, Javi, e tantos outros. Gracias, amigos.

Acknowledgements

My first thanks go to my family. To my parents, José and Lola, my sister Elba ("mom number two"), and my brother Juan. To my brother in law Tino, and my nephew and nieces, Oscar, Nuria and Lucía. For their love and support, but especially for putting up with me.

To Nieves, my director or Thesis and director of the Database Lab at the University of Coruña, for all her help in and out this thesis. To all the members, former and current, of the Lab: Jose, Eva, mon (in lowercase as you like it), Ángeles, Mariajo, Charo, Miguel, José Ramón (Viqueira), Antonio (Fari) e Tony. Thanks, and I hope I am not missing anybody. Also, special thanks to Dr. Héctor Hernández and the people of the Computer Science Department at New Mexico State University, where I started this Thesis.

Last, but not least, to all friend who were there when I needed them: Carlos, Ana, Manuel, Miryam, David, Luli, Javi, and so many more. Thanks, folks.

Contents

Chapter 1

Introduction

The main goal of database query optimization techniques is to transform a given query Q_1 into another query Q_2, which is semantically equivalent to Q_1 but can be more efficiently evaluated. Therefore, the problem of testing equivalence of queries is a fundamental issue for database query optimizers.

Two queries Q_1 and Q_2 are equivalent ($Q_1 \equiv Q_2$) if they obtain the same result when they are applied to the same database. That is,

$$Q_1 \equiv Q_2 \Longleftrightarrow \forall D,\ Q_1(D) = Q_2(D)$$

where $Q_1(D)$ and $Q_2(D)$ represent, respectively, the result of applying Q_1 and Q_2 to the database D. Note that, for the equivalence to hold, this result must be true for *any* ground database D.

Unfortunately, query equivalence in the general framework of relational algebra or calculus is undecidable [Tra50, Sol79, Fag93, AHV95]. However, there is a subclass of queries, the conjunctive queries, whose equivalence was shown to be decidable [CM77].

Conjunctive queries use only Selection, Projection and Cartesian product operators [Ull89]. They are the typical SQL queries used in commercial relational DBMSs, and they have been therefore extensively studied.

It is also possible to study conjunctive queries under the logic programming perspective [Llo87], which is the underlying data model of deductive databases. The most used query language in deductive databases is Datalog [CGT89], which is an adaptation of Prolog to deal with databases. Using deductive databases notation, a conjunctive query is a safe, nonrecursive Datalog rule. We shall see that the SQL and Datalog notations used to represent a conjunctive query are equivalent.

The test of equivalence of queries is usually done by checking their mutual containment:

$$Q_1 \equiv Q_2 \Longleftrightarrow Q_1 \leq Q_2 \wedge Q_2 \leq Q_1$$

where the containment of queries is expressed by the $\leq$ operator.

The containment of queries is defined as follows.

$$Q_1 \leq Q_2 \Longleftrightarrow \forall D,\ Q_1(D) \subseteq Q_2(D).$$

That is, a query Q_1 is contained in a query Q_2 if and only if the result of applying Q_1 to any ground database D is contained in the result of applying Q_2 to the same database D.

Most of the work that has been done to solve the conjunctive query containment problem (see, for example, [ASU79a, ASU79b, CM77]) assumes the set theoretic framework typical of the relational model [Cod70]. However, commercial database management systems use a bag theoretic semantics. Under bag semantics, duplicates of the tuples are allowed, and every table in a database is a "bag" or multiset of tuples. Any SQL query used in a commercial DBMS also produces as a result a bag of tuples, unless the `Distinct` clause is used in the query to remove duplicates.

The use of a different semantic framework has strong implications in the query containment/equivalence problem, because the results achieved in the set containment problem are not valid to test bag containment of queries. As we shall see, having two queries Q_1 and Q_2 such that Q_1 is contained in Q_2 under set semantics does not imply that Q_1 is contained in Q_2 under bag semantics, as Example 1.1 shows.

We use the symbols $\leq$ and $\equiv$ to denote, respectively, containment and equivalence of queries. However, in order to stress the differences among bag and set semantics, we shall use the s and b letters to denote containment and equivalence under set and bag semantics. Therefore, $Q_1 \leq_s Q_2$ would mean that the query Q_1 is contained in Q_2 under set semantics, and $Q_1 \equiv_b Q_2$ would mean that Q_1 and Q_2 are equivalent under bag semantics. These concepts and notations are formally defined in Chapter 2.

Example 1.1 Let D be a database with only one relation p with scheme $p(A, B)$. Let Q_1 and Q_2 be the following queries, represented using Datalog rules:

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

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

It is obvious that Q_1 is set contained in Q_2, because Q_1 is more restrictive. Assume that Q_1 obtains the fact $q(a, b)$, that is, $q(a, b) \in Q_1(D)$.

It means that the facts $p(a, b)$ and $p(b, a)$ are in D (they can be the same fact, if $a = b$). Since $p(a, b) \in D$, Q_2 also obtains the fact $q(a, b)$ (that is, $q(a, b) \in Q_2(D)$). This happens for any arbitrary fact $q(a, b)$ and any arbitrary database D, therefore $Q_1 \leq_s Q_2$.

However, under bag semantics, Q_1 is not bag contained in Q_2. We shall show it using a counterexample.

Let D be the following database:

p
a a
a a

That is, D has two copies of the fact $p(a, a)$.

In this case, Q_1 obtains 4 copies of the fact $q(a, a)$, because there are 4 different ways to obtain it, depending on which fact the predicates of Q_1 are mapped to:

- Both predicates are mapped to the first $p(a, a)$ fact.
- $p(X, Y)$ is mapped to the first fact and $p(Y, X)$ to the second one.
- $p(X, Y)$ is mapped to the second fact and $p(Y, X)$ to the first one.
- Both predicates are mapped to the second fact.

Q_2 obtains again a reproduction of the p relation into q, that is, 2 copies of the fact $q(a, a)$. Since Q_1 obtains 4 duplicates of the fact $q(a, a)$ and Q_2 obtains only 2, we can conclude that $Q_1 \not\leq_b Q_2$. □

On the other hand, the study of the query containment problem has been classically approached in a different way for queries with built-in predicates and queries without them. For example, Chandra and Merlin [CM77], Chaudhuri and Vardi [CV93] or Brisaboa and Hernández [BH97] dealt with conjunctive queries without built-in predicates, while Klug [Klu88] and Ullman [Ull89] explicitly dealt with conjunctive queries with built-in predicates. Built-in (also called interpreted) predicates are predicates of the form $X < Y$, $X \leq Y$, $X = Y$, or $X \neq Y$, where X and Y are either variables or constants in any ordered domain (not both constants). A conjunctive query can be an *equality* conjunctive query, if no built-in predicates are allowed, or an *inequality* conjunctive query if built-in predicates are allowed.

Note that we might consider equality queries having the equality comparison (=). However, we shall use a compressed form to write the queries.

This way, a built-in predicate $X = c$ where X is a variable and c is a constant, is reflected by replacing every of X by c in the query. If we have a built-in predicate $X = Y$, where both X and Y are variables, we will choose one of them as a representative and replace the other variable by the representative one. For example, a query like

$$q(X, Y, Z) \text{ :- } p(X, Y), p(T, Z), r(Z, W), Y = T, W = 4.$$

would be represented by the following (equivalent) query, without any built-in predicate:

$$q(X, Y, Z) \text{ :- } p(X, Y), p(Y, Z), r(Z, 4).$$

In this Thesis, we shall consider two factors that affect the problem of checking containment, and therefore equivalence, of conjunctive queries:

- *The underlying semantics:* The semantics can be set theoretic, where no duplicates of tuples are allowed, or bag (multiset) theoretic. Under bag semantics, tuples have an associated multiplicity, which indicates the number of copies of the tuple in the database.
- *Presence or absence of built-in predicates in the queries:* Inequality queries (that is, conjunctive queries with built-in predicates) will be treated differently than equality queries (without built-in predicates).

Considering these two factors, the containment problem can be studied under 4 different perspectives:

1. Set containment of equality queries,
2. Bag containment of equality queries,
3. Set containment of inequality queries, and
4. Bag containment of inequality queries.

The set containment of conjunctive queries was solved by Chandra and Merlin [CM77]; the bag containment of conjunctive queries was partially solved by Chaudhuri and Vardi [CV93] and then finally Brisaboa and Hernández [Bri97, BH97] offered a necessary and sufficient condition, along with a procedure, to test it. In these works, Brisaboa and Hernández transformed the problem of testing bag containment of equality queries into the problem of comparing two polynomials over $\mathbb{Z}^+$, the set of nonnegative integers.

There has also been work done in the set containment of inequality queries, but no definitive result has been achieved. The major contribution was made by Klug [Klu88], but he left the problem open when domain was nondense, like the integers. There has been other works in this type of containment [vdM97, IS97], but they either worked only for dense domains or lacked of a procedure. Finally, to the best of our knowledge, there has been no work done in the field of bag containment of inequality queries.

We shall present in this Thesis a general procedure to test conjunctive query containment, with or without built-in predicates, which works under set and bag semantics. This procedure, denoted *QCC* (Query Containment Checker) is based on the idea of using a finite set of databases built from the body of the query Q_1, denoted Canonical Database Set ($CDBS(Q_1)$) [Bri97], to test the containment.

1.1 Overview of the Thesis

This work tries to offer a unified procedure, *QCC* (Query Containment Checker) to test, in three steps, whether a conjunctive query Q_1 is contained into another conjunctive query Q_2. This procedure tests the containment using only a small and finite set of databases. This set, denoted $CDBS(Q_1)$ (Canonical Database Set for a query Q_1) is built algorithmically from the body of Q_1, using a procedure derived from the one described in [BH97]. This thesis proves that, for two conjunctive queries Q_1 and Q_2, either equality or inequality queries, under set or bag semantics,

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

That is, the containment can be checked using only a finite and usually small number of databases, instead of using the infinite number of ground databases from which Q_1 and Q_2 could derive new facts (which would be, of course, impossible to test).

The use of this procedure reduces the conjunctive query containment problem to different problems, depending on the underlying semantics and the presence or absence of built-in predicates:

- The application of *QCC* to test set containment of equality queries is reduced to the problem of finding an assignment mapping, in a similar way as Chandra and Merlin originally solved the problem in [CM77].
- The application of *QCC* to test the bag containment of equality queries reduces to a comparison of two polynomials. We will show that the procedure described in [BH97] perfectly fits into the 3 steps that *QCC* follows.

- The application of QCC to test the set containment of inequality queries (a preliminary version of the work presented in Chapter 6 in this Thesis was published in [BHPP98]) reduces the problem to the test of the unsatisfiability of a formula composed of equalities and inequalities.

- Finally, the application of QCC to test bag containment of inequality queries is reduced to the test of the unsatisfiability of a formula plus the comparison of pairs of polynomials.

The general layout of this Thesis is the following.

In Chapter 2, conjunctive queries are defined, as well as how to apply them to a database under set or bag semantics.

One of the main contribution of this Thesis is shown in chapter 3, where a general procedure to test the containment of conjunctive queries is described. This procedure, generally called QCC, will be specifically described for the 4 types of containment considered in this Thesis, proving its correctness.

The next part of this Thesis deals with set semantics. Chapter 4 shows the previous work about set containment of equality as well as inequality queries. Chapter 5 describes how QCC is adapted to test set containment of equality queries, and Chapter 6 shows the use of QCC to test set containment of inequality queries. Both chapters constitue an original contribution of this Thesis to the problem of testing set containment of conjunctive queries.

The last chapters deal with bag semantics. Chapter 7 shows the previous work about the bag containment of conjunctive queries. Chapter 8 describes the adaptation of QCC to test bag containment of equality queries, using a method derived from the one described in [Bri97] which fits the three steps of QCC. Chapter 9 shows the application of QCC to test bag containment of inequality queries, and this is again a new, original contribution of this Thesis to the problem of testing containment among conjunctive queries. The correctness of the procedure is also demonstrated in each of these chapters.

Finally, Chapter 10 shows our conclusions.

Chapter 2

Definition of conjunctive queries

2.1 Introduction

Conjunctive queries are the most common SQL queries used in the commercial database management systems, and have been, therefore, extensively studied [CM77, ASU79a, ASU79b, Ull82, Klu88, Ull89, IR92, CV93, BH97, BHPP98].

In this chapter, we will define the concept of conjunctive queries, distinguishing two types, equality and inequality queries, because the difference between these types has a strong impact in how our general procedure to test conjunctive query containment works.

A different concept, which also has implications on the query containment problem, is the underlying (set or bag theoretic) semantics. The application of a query to a database, as well as the concepts of containment and equivalence of queries, under both semantics, are also defined in this chapter.

2.2 Conjunctive queries

A *conjunctive query*, under relational algebra theory, is a query that uses only Selection, Projection and Cartesian product operations. Using deductive databases notation, a conjunctive query is a safe, nonrecursive rule with the predicates of the body defined over EDB (Extensional Databases) predicates [Ull89].

Conjunctive queries can be represented using different notations that are equivalent. Among these notations, the most common are Datalog rules (deductive databases notation), SQL queries, or Relational Algebra expressions. Due to its simplicity and easiness of use, we shall use Datalog rules to represent conjunctive queries. After formally defining conjunctive queries, we shall prove that the SQL and Datalog notations are equivalent.

Depending on the presence or absence of built-in predicates in the bodies of the rules, we distinguish two types of conjunctive queries:

Equality queries: They are conjunctive queries where built-in predicates are not allowed. The general form of an equality query is

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

where

- $q(\vec{X})$ is the *query predicate*, being $\vec{X}$ a vector or tuple of variables.
- Every $p_i(\vec{Y_i})$ is an *ordinary predicate* defined over EDB predicates, having p_i as its predicate name, and being $\vec{Y_i}$ a vector or tuple of constants or variables.

Inequality queries: They are conjunctive queries with *built-in predicates*. 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 $\theta \in \{=, \neq, <, \leq, >, \geq\}$.

As stated before, the representations of a conjunctive query in SQL and Datalog notations are equivalent. The following algorithm transforms a conjunctive query written in SQL into its equivalent Datalog notation [Ull89].

Algorithm 1 Transform an SQL conjunctive query into its equivalent Datalog rule.
Input: An SQL query of the form

SELECT DISTINCT A_1, ..., A_n
FROM $table_1$, ..., $table_t$
WHERE $< condition_1 > and \ldots and < condition_c >$

Output: The same query written as a Datalog rule.
Method:

1. There will be a different variable for each attribute in the relation scheme of all tables in the FROM clause.

2. For each of the relations in the FROM clause, add a predicate to the body of the rule, with the same number of attributes and in the same order as in the relation scheme.

3. For each of the equalities or inequalities in the WHERE clause, add a built-in predicate to the body of the rule that establishes the (in)equality between two variables or one variable and one constant.

4. The variables in the head of the rule are those variables that appear in the SELECT clause.

Note that this algorithm produces a Datalog rule with explicit equalities, that is, if $X = Y$, then this built-in is added to the rule. A "compressed" form of the rule can be built just considering all variables related by an $=$ operator as an equivalence class, replacing all the variables by the representative of the equivalence class and removing the equality from the body of the rule. In the same way, if there is an equality $X = c$, being X a variable and c a constant, every occurrence of X is replaced by c. □

Example 2.1 Let D be a database with the following scheme:

$emp(EmpNo, EmpName, DeptNo, Salary)$

$dept(DeptNo, DeptName)$

The query "Obtain the names of the employees working in the 'Sales' department that earn more than 14000" can be represented as the following SQL query:

```
SELECT DISTINCT EmpName
    FROM emp, dept
    WHERE emp.DeptNo = dept.Deptno
    AND dept.Deptno = "Sales"
    AND emp.Salary > 14000
```

Following the previous algorithm, the equivalent Datalog rule is built:

- There are the following variables:

$$EmpNo, EmpName, EmpDeptNo, Salary, DeptNo, DeptName$$

- The predicates in the body of the rule are:

$$emp(EmpNo, EmpName, EmpDeptNo, Salary),$$

$$dept(DeptNo, DeptName)$$

- The following built-in predicates are added.

$$EmpDeptNo = Deptno, \quad Deptno = "Sales", \quad Salary > 14000$$

- The variables in the head of the rule are:

$$Empname$$

Therefore, the Datalog rule that represents this query is

$$\begin{aligned} result(EmpName) \text{ :- } & emp(EmpNo, EmpName, EmpDeptNo, Salary), \\ & dept(DeptNo, DeptName), \\ & EmpDeptNo = DeptNo, DeptName = "Sales", \\ & Salary > 14000 \end{aligned}$$

We have the equality $DeptName = "Sales"$, therefore $DeptName$ can be replaced by $"Sales"$ in the query. Additionally, the variables $EmpDeptNo$ and $DeptNo$ are in the same equivalence class. Considering $DeptNo$ as the representative of this class, the query can be rewritten in a compressed form as

$$\begin{aligned} result(EmpName) \text{ :- } & emp(EmpNo, EmpName, DeptNo, Salary), \\ & dept(DeptNo, "Sales"), \\ & Salary > 14000 \end{aligned}$$

□

It is obvious, by how Algorithm 1 works, that the SQL query and the Datalog rule are equivalent.

The application of an SQL query to a database in order to obtain new facts or tuples works in a different (although equivalent) way than the same query written as a Datalog rule. The execution of an SQL query follows 3 steps:

- Compute the Cartesian Product of all the relations in the FROM clause.
- Select the tuples that satisfy the conditions expressed in the WHERE clause from the previous result.

- Project only those attributes that appear in the SELECT clause.

The application of a query written as a Datalog rule uses assignment mappings to derive new facts. An *assignment mapping* [Ull82] 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_1 be a query of the form

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

and D any arbitrary database. Assume there is an assignment mapping τ from Q_1 to D. Then,

- every $\tau(p_i(\vec{Y_i}))$ $(1 \leq i \leq n)$ is a fact in D.
- every $\tau(F_i)$ $(1 \leq i \leq k)$ is *true*.
- $\tau(q(\vec{X}))$ is the derived fact.

Example 2.2 Let us use the query from Example 2.1. Let us apply it to the following database D.

emp			
EmpNo	EmpName	DeptNo	Salary
10	John Smith	10	11000
20	Peter Sellers	10	29000
30	Joe Sand	20	13000
40	Mary Raines	30	25000

dept	
DeptNo	DeptName
10	Sales
20	Accounting
30	Marketing

Let us apply the SQL query

```
SELECT DISTINCT EmpName
    FROM emp, dept
    WHERE emp.DeptNo = dept.Deptno
    AND dept.Deptno = "Sales"
    AND emp.Salary > 14000
```

- Compute the Cartesian Product of the tables in the FROM clause (*emp* and *dept*):

Cartesian Product					
emp.EmpNo	emp.EmpName	emp.DeptNo	emp.Salary	dept.DeptNo	dept.DeptName
10	John Smith	10	11000	10	Sales
10	John Smith	10	11000	20	Accounting
10	John Smith	10	11000	30	Marketing
20	Peter Sellers	10	29000	10	Sales
20	Peter Sellers	10	29000	20	Accounting
20	Peter Sellers	10	29000	30	Marketing
30	Joe Sand	20	13000	10	Sales
30	Joe Sand	20	13000	20	Accounting
30	Joe Sand	20	13000	30	Marketing
40	Mary Raines	30	25000	10	Sales
40	Mary Raines	30	25000	20	Accounting
40	Mary Raines	30	25000	30	Marketing

- Select the tuples that satisfy the constraints in the WHERE clause:

Selection of tuples					
emp.EmpNo	emp.EmpName	emp.DeptNo	emp.Salary	dept.DeptNo	dept.DeptName
20	Peter Sellers	10	29000	10	Sales

- Project the attributes in the SELECT clause:

Projection of attributes
emp.EmpName
Peter Sellers

Let us apply now the Datalog rule that represents the same query:

$$result(EmpName) \text{ :- } emp(EmpNo, EmpName, DeptNo, Salary),$$
$$dept(DeptNo, \text{"}Sales\text{"}),$$
$$Salary > 14000$$

There is only one assignment mapping τ from this query to D:

$\tau(EmpNo) = 20;$

$\tau(EmpName) = \text{"}Peter\ Sellers\text{"};$

$\tau(DeptNo) = 10;$

$\tau(Salary) = 29000;$

$\tau(\text{"}Sales\text{"}) = \text{"}Sales\text{"}$

That is,

$$\tau(emp(EmpNo, EmpName, DeptNo, Salary)) = \\ emp(20, \text{"}Peter\ Sellers\text{"}, 10, 29000);$$

$$\tau(dept(DeptNo, \text{"}Sales\text{"})) = dept(10, \text{"}Sales\text{"})$$

Therefore, the result of applying this query is

$$\tau(result(EmpName)) = result(\text{"}Peter\ Sellers\text{"})$$

Obviously, the results obtained by the SQL query and the Datalog rule are the same[1]. □

2.3 Set and Bag Semantics

As shown in Example 1.1, set containment of conjunctive queries does not imply bag containment (the opposite is true: bag containment does imply set containment). The fundamental concept, which makes this difference, is the multiplicity of the facts that appears under bag semantics.

We shall describe in this section how relations in databases are represented in both frameworks, as well as how to apply a query to a database. The formal definitions of set and bag containment are also given in this section.

2.3.1 Definitions under set semantics

Under the set theoretic framework, both relations (tables) in a database and the result of applying a query to a database 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 $p(A_1, \ldots, A_l)$.

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, $Q_1(D) \subseteq_s Q_2(D)$, that is, the set of facts obtained by Q_1 is a subset of the set of facts obtained by Q_2:

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

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.

[1]Note that they are equivalent under set semantics, because Datalog always removes duplicates of the facts, but under SQL it must be done explicitly by using the `Distinct` clause.

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 $Q_2 \leq_s Q_1$.

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

2.3.2 Definitions under bag semantics

Under bag semantics, a relation in a database is a *bag* or multiset of facts, where every fact has a number of copies in the relation. It can be seen as if every fact has an associated integer that indicates its *multiplicity*. Under this point of view, any relation is a set of elements of the form $p(A_1, \ldots, A_l; [m])$. For each of these elements, p is a predicate name, $A_1, \ldots, A_l$ are constants of the domain, and m is the *multiplicity* or number of copies of the fact $p(A_1, \ldots, A_l)$ in the database. If a fact is not in a database D, then its multiplicity in D is 0.

The multiplicity of a fact in a database D is represented as $|p(A_1, \ldots, A_l)|_D = m$.

Example 2.3 Let D be a database with only one relation named *menu*. The scheme for this relation is $menu(Firstdish, Seconddish)$, and it represents the menus ordered for dinner in a given restaurant in an evening. An instance of this database could be:

menu
soup, beefsteak; [3]
salad, burrito [9]

That means that there were 3 people that had for dinner soup and then a beefsteak, and 9 people who had a salad and a burrito. The multiplicity of the first fact is represented as

$$|menu(soup, beefsteak)|_D = 3.$$

There were no people who had lasagna and octopus for dinner, therefore

$$|menu(lasagna, octopus)|_D = 0.$$

□

Definition of subbag

The concept of subbag will be fundamental to test the 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:

$$B \subseteq_b B' \Longleftrightarrow \forall t \in B,\ |t|_B \leq |t|_{B'}$$

The equality among subbags, represented by $=_b$, can be checked via their mutual containment.

$$B =_b B' \iff B \subseteq_b B' \wedge B' \subseteq_b B$$

Example 2.4 Let D, D' and D'' be the following bags of tuples:

D
soup, beefsteak; [3]
salad, burrito; [9]

D'
soup, beefsteak; [4]
salad, burrito; [9]

D''
soup, beefsteak; [2]
salad, burrito; [10]
salad, pizza; [4]

It is obvious that $D \subseteq_b D'$. But $D \not\subseteq_b D''$, even when D'' has more tuples than D. This happens because there is a fact, $menu(soup, beefsteak)$, with more multiplicity in D than in D''. □

Union of bags

The union of two bags B and B', represented $B \cup B'$, is defined as

$$B \cup B' = \{(t;[m]) \mid t \text{ is in } B \text{ or } t \text{ is in } B', \text{ and } m = |t|_B + |t|_{B'}\}$$

Definition of bag containment of conjunctive queries

The containment of conjunctive queries under bag semantics, also denoted 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:

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

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

Definition of bag equivalence of conjunctive queries

As for the case of set equivalence, query equivalence is defined by mutual inclusion. 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 $Q_2 \leq_b Q_1$.

$$Q_1 \equiv_b Q_2 \iff Q_1 \leq_b Q_2 \wedge Q_2 \leq_b Q_1$$

Computing the multiplicity of a derived fact

In order to test the bag containment of queries, we must know how to compute the multiplicity of a fact derived by a query when it is applied to a database.

Given a database D and a query Q, $Q(D)$ represents the derived facts obtained by applying Q to D, and it is also a bag of facts. The multiplicity of a fact t in $Q(D)$ will be represented as $|t|_{Q(D)}$, and it is computed as follows.

Let Q_1 be a query of the form $q(\vec{X})$:- $p_1(\vec{Y_1}), \ldots, p_n(\vec{Y_n})$, and let D be a database. Assume there are l assignment mappings $\tau_1, \ldots, \tau_l$ from Q to D that obtain the new derived 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, which has a multiplicity. Let us represent this multiplicity 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:

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

Let us show it through an example.

Example 2.5 Let Q_1 be $q(X)$:- $p(X, Y), p(Y, Z)$. Let D be the database

p
a b; [3]
b c; [5]
b d; [4]

There are two mappings τ_1 and τ_2 that obtain the fact $q(a)$:

$$\tau_1(X) = a; \tau_1(Y) = b; \tau_1(Z) = c$$

$$\tau_2(X) = a; \tau_2(Y) = b; \tau_2(Z) = d$$

The multiplicities obtained by each mapping are

$$m_1 = |p(a,b)|_D \times |p(b,c)|_D = 3 \times 5 = 15$$

$$m_2 = |p(a,b)|_D \times |p(b,d)|_D = 3 \times 4 = 12$$

and the total multiplicity is

$$|q(a)|_{Q_1(D)} = m_1 + m_2 = 15 + 12 = 27.$$

□

2.4 Summary

We have defined in this chapter the necessary concepts to tackle the problem of containment of conjunctive queries under the 4 perspectives shown in the introduction. Therefore, the concepts of equality and inequality queries have been defined. Set and bag semantics have also been described, as well as how to apply a query to a database under either of them.

Chapter 3

QCC: Query Containment Checker

3.1 Introduction

The *Query Containment Checker* (QCC) is a general procedure that can decide 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 the same for the 4 cases of conjunctive query containment covered by this Thesis (under set or bag semantics, for equality or inequality queries), but there will be some particularization for each specific case.

Given that the first step is to build $CDBS(Q_1)$, we shall begin describing the canonical database set for a query Q_1.

The idea under canonical databases [BH97] is to capture all the assignment mappings that can be applied from a query Q to any database in order to obtain a new fact. The *Canonical Database Set* for a query Q ($CDBS(Q)$) is a finite set of databases with uninterpreted constants in its facts. It captures all the patterns of equalities and inequalities among the constants of a database D where the variables of Q are mapped when Q is applied to D.

The algorithm that builds $CDBS(Q)$ can be conceptually divided into two steps. The first step, which is common for the 4 classes of conjunctive query containment covered in this Thesis, is to build the canonical databases. These canonical databases are adapted in the second step, so they are suitable to test a specific type of conjunctive query containment. This particularization is made by adding some constraints to the canonical databases (for the containment of inequality queries) or assigning a symbolic multiplicity to the facts in the databases (if the underlying semantics is bag theoretic).

The common part of the algorithm is described in this chapter, and the particularization needed by each type of containment will be described in chapters 5, 6, 8, and 9, where the different types of the containment problem are described. The last part of this chapter describes the 3 steps of QCC.

3.2 Preliminary definitions

Let Q be a conjunctive query of the form

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

Note that Q has built-in predicates, that is, Q is an inequality query. We shall use this form of conjunctive query to describe the following concepts because it is more general, since an equality query is just an inequality query without built-in predicates.

Using the predicates in the body of Q, the following concepts are defined:

- $db(Q)$: is the set of ordinary predicates in the body of Q.

  $$db(Q) = \{p_i(\vec{V}) \mid p_i(\vec{V}) \text{ is an ordinary predicate in the body of } Q\}$$

  Recall that an ordinary predicate is defined over an EDB predicate. Built-in predicates are not included in $db(Q)$.

- V_Q: is an ordering $\langle V_1, \cdots, V_q \rangle$ of the set of all the variables that appear in the predicates in $db(Q)$.

- A_Q: is a set of q new, different uninterpreted constants, q being the cardinality of V_Q.

  $$A_Q = \{A_1, \ldots, A_q \mid \forall i, j, (1 \leq i \neq j \leq q)\ A_i \neq A_j\}$$

- *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 let $p_i(Y_1, ..., Y_{r_i})$ be a predicate in Q. We define the application of θ to $p_i(Y_1, ..., Y_{r_i})$ as the fact $p_i(\theta(Y_1), ..., \theta(Y_{r_i}))$.

- *Isomorphic Q-mappings:* Two Q-mappings $\theta_i = (A_{i_1}, ..., A_{i_q})$ and $\theta_j = (A_{j_1}, ..., A_{j_q})$ are *isomorphic* if there are two mappings γ_1 and γ_2 (from A_Q to A_Q) such that $(\gamma_1(A_{i_1}), ..., \gamma_1(A_{i_q})) = \theta_j$, and $(\gamma_2(A_{j_1}), ..., \gamma_2(A_{j_q})) = \theta_i$. In other words, two Q-mappings are isomorphic if the q-tuples that represent them are identical after a consistent renaming of their uninterpreted constants. Isomorphic Q-mappings are used to define a minimal number of canonical databases.

- 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 uninterpreted constants. All the uninterpreted constants must be different $(A_j \neq A_k, \forall j, k\ 1 \leq j \neq k \leq q)$).

  Each of these canonical databases will be adapted to test a specific type of conjunctive query containment.

- *Canonical fact t_{d_i}:*

  Let Q be the query $q(\vec{X})$:- $p_1(\vec{Y_1}), \ldots, p_l(\vec{Y_l}), K_1, \ldots, K_n$, and d_i be the canonical database obtained by applying the Q-mapping θ_i to $db(Q)$. Then, the canonical fact t_{d_i} is the fact $\theta_i(q(\vec{X}))$.

The following example illustrates these definitions.

Example 3.1 Let us consider the following query Q:

$$Q\ :\ q(X, Y, Z)\ \text{:-}\ r(X, U), r(U, Z), p(U, Y).$$

For this query,

- $db(Q) = \{r(X, U), r(U, Z), p(U, Y)\}$.
- $V_Q = \langle X, Y, Z, U\rangle$ is an ordering of its variables.
- $A_Q = \{A_1, A_2, A_3, A_4\}$ is a set of 4 new, different uninterpreted constants.

Let θ_1 be the Q-mapping (A_4, A_3, A_3, A_4); θ_1 denotes the mapping $\theta_1(X) = \theta_1(U) = A_4$ and $\theta_1(Y) = \theta_1(Z) = A_3$. Applying the mapping θ_1 to $db(Q)$, we obtain the canonical database

$$\theta_1(db(Q)) = \{r(A_4, A_4), r(A_4, A_3), p(A_4, A_3)\}$$

Another Q-mapping, which is isomorphic to θ_1, is $\theta_2 = (A_1, A_2, A_2, A_1)$.

The canonical fact that corresponds to the Q-mapping θ_1 is $t_{d_1} = q(A_4, A_3, A_3)$. □

3.3 Building the canonical database set for a query

This section offers an algorithm to build all the non isomorphic canonical databases d_i for a conjunctive query Q_1. However, before formally describing the algorithm, let us show how it works through an example.

Example 3.2 Let the query Q_1 be $q(X, Y, Z) :- r(X, U), r(U, Z), p(U, Y)$. The letters A, B, C, D (which represent uninterpreted constants) will be used to identify the 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, and so on.

However, using all the 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 3.1 is generated.

Table 3.1: Canonical databases generated for case 1

NAME	$\theta_i(db(Q_1))$		MAPPING	t_{d_i}
	r	p		q
θ_1	(A, D) (D, C)	(D, B)	$\theta_1(X) = A$; $\theta_1(Y) = B$; $\theta_1(Z) = C$; $\theta_1(U) = D$	ABC

Case 2: Three variables are mapped to the same value, the other one is mapped to a different value. Table 3.2 shows the 4 canonical databases for this case.

Table 3.2: Canonical databases generated for case 2

NAME	$\theta_i(db(Q_1))$		MAPPING	t_{d_i}
	r	p		q
θ_2	(A,B) (B,A)	(B,A)	$\theta_2(X) = \theta_2(Y) = \theta_2(Z) = A;\ \ \theta_2(U) = B$	AAA
θ_3	(A,A) (A,B)	(A,A)	$\theta_3(X) = \theta_3(Y) = \theta_3(U) = A;\ \ \theta_3(Z) = B$	AAB
θ_4	(A, A)	(A, B)	$\theta_4(X) = \theta_4(Z) = \theta_4(U) = A;\ \ \theta_4(Y) = B$	ABA
θ_5	(B, A) (A, A)	(A, A)	$\theta_5(Y) = \theta_5(Z) = \theta_5(U) = A;\ \ \theta_5(X) = B$	BAA

Case 3: Two variables are mapped to the same value and the other two are mapped to another value. For this case, 3 canonical databases are generated, as shown in Table 3.3.

Table 3.3: Canonical databases generated for case 3

NAME	$\theta_i(db(Q_1))$		MAPPING	t_d
	r	p		q
θ_6	(A,B) (B,B)	(B,A)	$\theta_6(X) = \theta_6(Y) = A;\ \ \theta_6(Z) = \theta_6(U) = B$	AAB
θ_7	(A, B) (B, A)	(B, B)	$\theta_7(X) = \theta_7(Z) = A;\ \ \theta_7(Y) = \theta_7(U) = B$	ABA
θ_8	(A, A) (A, B)	(A, B)	$\theta_8(X) = \theta_8(U) = A;\ \ \theta_8(Y) = \theta_8(Z) = B$	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 3.4.

Case 5: The four variables are mapped to the same value. Only one canonical database, shown in Table 3.5, is generated in this case.

We have generated 15 canonical databases for Q_1. Notice that not all the possible Q-mappings are required. For example, the Q-mapping of X and Y to B and U and Z to D would produce a database that has the same

Table 3.4: Canonical databases generated for case 4

NAME	$\theta_i(db(Q_1))$		MAPPING	t_{d_i}
	r	p		q
θ_9	(A,C) (C,B)	(C,A)	$\theta_9(X) = \theta_9(Y) = A$; $\theta_9(Z) = B$; $\theta_9(U) = C$	AAB
θ_{10}	(A,C) (C,A)	(C,B)	$\theta_{10}(X) = \theta_{10}(Z) = A$; $\theta_{10}(Y) = B$; $\theta_{10}(U) = C$	ABA
θ_{11}	(A,A) (A,C)	(A,B)	$\theta_{11}(X) = \theta_{11}(U) = A$; $\theta_{11}(Y) = B$; $\theta_{11}(Z) = C$	ABC
θ_{12}	(B,C) (C,A)	(C,A)	$\theta_{12}(Y) = \theta_{12}(Z) = A$; $\theta_{12}(X) = B$; $\theta_{12}(U) = C$	BAA
θ_{13}	(B,A) (A,C)	(A,A)	$\theta_{13}(Y) = \theta_{13}(U) = A$; $\theta_{13}(X) = B$; $\theta_{13}(Z) = C$	BAC
θ_{14}	(B,A) (A,A)	(A,C)	$\theta_{14}(Z) = \theta_{14}(U) = A$; $\theta_{14}(X) = B$; $\theta_{14}(Y) = C$	BCA

Table 3.5: Canonical databases generated for case 5

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

pattern of equalities as $\theta_6(db(Q_1))$ (it would be isomorphic to $\theta_6(db(Q_1))$). With four variables there are 256 different possible canonical databases, but there are only 15 different (non isomorphic) ones. In [Bri97], the reader can find 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. □

3.4 Algorithm to build $CDBS(Q)$

The following algorithm builds the set of canonical databases for a conjunctive query Q. This algorithm generates all the non isomorphic Q-mappings, which will be used to build all the canonical databases d_i in $CDBS(Q_1)$. Since the canonical databases must be adapted to test each type of containment, the last part of the algorithm is a call to another algorithm, which will be described in the corresponding chapter.

The set of all non isomorphic canonical databases that can be built from a query Q is denoted $CDBS(Q)$ (canonical database set of Q):

$$CDBS(Q) = \{\theta_1(db(Q)), \ldots, \theta_x(db(Q))\}$$

where x is the number of nonisomorphic Q-mappings that can be generated from the body of Q.

Algorithm 2 Algorithm to build $CDBS(Q)$.

Input: $Q = q(\vec{X}) : -p_1(\vec{Y_1}), ..., p_l(\vec{Y_l}), K_1, \ldots, K_n$.
Output: $CDBS(Q)$
Method:

1. Initialization.
   Let $db(Q) = \{p_1(\vec{Y_1}), ..., p_l(\vec{Y_l})\}$;
   Let $V_Q = \langle V_1, ..., V_q \rangle$ be an ordering of all the variables that appear in $db(Q)$;
   Let $A_Q = \{A_1, ..., A_q\}$ be q new, distinct (uninterpreted) constants
   ($A_i \neq A_j$, if $i \neq j, 1 \leq i, j \leq q$);
   Let $j = 1$;
   Let $Mappings = \emptyset$;
2. Definition of Q-mappings.

```
for i_1 = 1 to q
  for i_2 = 1 to q
    ...
      for i_q = 1 to q {
        θ_j = (A_{i_1}, A_{i_2}, ..., A_{i_q});
        if (there is no Q-mapping in Mappings that is isomorphic to θ_j){
          Mappings = Mappings ∪ {θ_j};
          j = j + 1;
        }
      }
```

3. Generation of $CDBS(Q)$

```
for i = 1 to j − 1
  Generate θ_i(db(Q))
  //CDBS(Q) = {θ_1(db(Q)), ..., θ_{j−1}(db(Q))}
```

4. Call the algorithm to adapt $CDBS(Q)$ for each type of containment.
   (This step is shown individually for each type of containment)

□

3.5 Interest of canonical databases

The use of canonical databases in the query containment problem offers an important advantage: it reduces the problem of checking the containment over the infinite possible number of ground databases from which Q_1 and Q_2 can derive new facts to check it over a finite (usually small) set of databases, the canonical databases.

QCC is based on the use of canonical databases due to a fundamental property of $CDBS(Q_1)$: 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 $CDBS(Q_1)$. This will be proven for each type of conjunctive query containment, because canonical databases will be different and the isomorphism must consider different properties of the canonical databases for each specific case (multiplicities in the facts and/or constraints in the databases). We shall prove it for each type of conjunctive query containment, but all the proofs take advantage of the way the canonical databases are built. Given that $CDBS(Q_1)$ covers all the possible patterns of equalities among uninterpreted constants, the subset of D where the atoms in Q_1 are mapped by an assignment mapping from Q_1 to D will be isomorphic to a canonical database. The following lemma offers a preliminary proof of this fact.

Lemma 3.1 *Let Q_1 be the conjunctive query*

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

Let τ be an assignment mapping from Q_1 to a database D. Let $sd = \{\tau(p_1(\vec{Y_1})), \cdots, \tau(p_l(\vec{Y_l}))\}$; i.e., sd is the subset of D on which τ maps the ordinary predicates of Q_1. Then sd is isomorphic to a canonical database $d \in CDBS(Q_1)$.

Proof:

The set sd is the subset of D obtained by applying τ to the body of Q_1. Note that every canonical database d is obtained by using a mapping from $db(Q_1)$, which is isomorphic to the body of Q_1, to a set of uninterpreted constants A_Q.

Assume that τ maps every variable of Q_1 to the same constant a in sd. By construction of $CDBS(Q_1)$, there exists a canonical database, say $d_1 = \theta_1(db(Q_1))$, where the Q-mapping θ_1 maps every variable of Q to the same uninterpreted constant, $A \in A_Q$. It is obvious that sd and d_1 are isomorphic, because if, in every fact $p_i(A, \ldots, A)$ of d_1, we replace A by a, sd and d_1 are identical.

Now, assume that τ maps all variables of Q_1 to the constant a, except one, which is mapped to a different constant b. As in the previous case, there exists a canonical database built using a Q-mapping with the same pattern of equalities among the uninterpreted constants. Therefore, there will be a canonical database, say d_2, which is isomorphic to sd for this case.

The same method of reasoning can be used to cover all possible patterns of equalities among the constants in sd to which the variables of Q_1 are mapped. By construction of $CDBS(Q_1)$, the equalities among the uninterpreted constants in the facts of the canonical databases cover all the possible patterns of equalities among the variables of Q_1 when they are mapped to any ground database D. Therefore, there exists always a canonical database d_i isomorphic to sd. $\square$

3.6 General procedure to test query containment

QCC, the general procedure to test whether a conjunctive query Q_1 is contained in a conjunctive query Q_2 consists of the following three steps.

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

$CDBS(Q_1)$ is built using Algorithm 2. The last step of this algorithm 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, by construction of $CDBS(Q_1)$, Q_1 always obtains the canonical fact.

Step 3: Check the containment.

The 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 that QCC is that these three steps are the same for 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 to test the conjunctive query containment. This is the major contribution of this Thesis.

3.7 Summary

This chapter has shown one of the main contribution of this Thesis, the QCC procedure. QCC is a procedure that consits on three steps, and it can be used to test set or bag containment of equality as well as inequality queries.

The first step of this procedure, the construction of $CDBS(Q_1)$, has been shown in more detail, specifying an algorithm (adapted from [Bri97]) that builds the initial set of canonical databases that will be used in the next steps to test the containmet. The adaptation of the canonical databases in $CDBS(Q_1)$ to test each specific case of query containment will be described in the corresponding chapter.

Chapter 4

Previous work about set containment of conjunctive queries

4.1 Introduction

This chapter presents the work that has been done in the field of set containment of equality and inequality conjunctive queries.

The set containment of equality queries was fully solved by Chandra and Merlin [CM77], using the concept of containment mapping. However, the set containment of inequality queries was not fully solved. There has been extensive work on it (see [Klu88, Ull89, vdM97, IS97]), but the proofs given by these authors either lack a procedure or are applicable only when the underlying domain is dense.

4.2 Set containment of equality queries

We shall use the following general form to represent two equality queries Q_1 and Q_2:

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

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

The definitive contribution to the containment problem for equality queries under set semantics was made by Chandra and Merlin [CM77]. They used the concept of *containment mapping* to prove the query containment.

A *containment mapping* γ from a query Q_2 to a query Q_1, as defined in [CM77], is a mapping from the symbols of Q_2 to those of Q_1 such that:

- The mapping γ is the identity for constants and predicate names.
- It must map the head of Q_2 to the head of Q_1: $\gamma(q(\vec{V})) = q(\vec{W})$.
- Every atom in the body of Q_2 must be mapped to an atom in the body of Q_1: $\forall i\ \exists j\ (1 \leq i \leq k, 1 \leq j \leq l)\ \ \gamma(p_i(\vec{Z_i})) = p_j(\vec{Y_j})$. Note that it is not necessary that every atom of Q_1 is reached by an atom in Q_2.

The following theorem [CM77] states a necessary and sufficient condition for the containment of equality queries.

Theorem 4.1 *Let Q_1 and Q_2 be two equality queries. Then, $Q_1 \leq_s Q_2$ if and only if there exists a containment mapping γ from Q_2 to Q_1:*

$$Q_1 \leq_s Q_2 \Longleftrightarrow \exists\gamma,\ \gamma \textit{ is a containment mapping } Q_2 \rightarrow Q_1$$

Proof: (adapted from [Bri97])

IF: Assume there is a containment mapping γ from Q_2 to Q_1, and consider any arbitrary ground database D. Q_1 derives a fact t from D if and only if there exists an assignment mapping τ from Q_1 to D that maps every predicate in the body of Q_1 to a fact in D. That is, every $\tau(p_i(\vec{Y_i}))$ is a fact in D, and $t = \tau(q(\vec{W}))$.

Now, consider the consecutive application of γ and τ to Q_2. Applying them to the head of Q_2, we obtain $\tau(\gamma(q(\vec{V}))) = \tau(q(\vec{W})) = t$.

The application of $\tau \circ \gamma$ to every predicate in the body of Q_2 is always possible, since $\gamma(p_i(\vec{Z_i})) = p_j(\vec{Y_j})$, for some j $(1 \leq j \leq l)$. Then, $\tau(\gamma(p_i(\vec{Z_i}))) = \tau(p_j(\vec{Y_j}))$, which is a fact in D. Therefore, every fact t derived by Q_1 using the assignment mapping τ is also derived by Q_2, using the assignment mapping $\tau \circ \gamma$. Thus, $Q_1 \leq_s Q_2$.

ONLY IF: Assume $Q_1 \leq_s Q_2$. We want to prove that there is a containment mapping γ from Q_2 to Q_1.

Let us build a ground database D, defining an assignment mapping τ that maps every atom in the body of Q_1 to a different fact in D. That is,

$$\tau(p_1(\vec{Y_1})), \ldots, \tau(p_l(\vec{Y_l})) \in D$$

Q_1 obtains the fact $t = \tau(q(\vec{W}))$ from D. Since $Q_1 \leq_s Q_2$, Q_2 also obtains the same fact. Let λ be the assignment mapping from Q_2 to D that obtains it:

$$\lambda(q(\vec{V})) = t = \tau(q(\vec{W}))$$

Likewise, every predicate in the body of Q_2 must be mapped to a fact in D by the assignment mapping λ:

$$\forall i \ (1 \leq i \leq k) \ \exists j \ (1 \leq j \leq l), \ \lambda(p_i(\vec{Z_i})) = \tau(p_j(\vec{Y_j}))$$

It is clear that applying λ to the atoms of Q_2 followed by the application of the inverse function of the assignment mapping τ (denoted τ^{-1}), we obtain the atoms of Q_1. Therefore, the containment mapping γ from Q_2 to Q_1 we are looking for is

$$\gamma = \tau^{-1} \circ \lambda.$$

□

With this result, the set containment of equality queries was fully solved. More information can be found in [CM77] and [Ull82].

4.3 Set containment of inequality queries

The following theorem [Ull89] provides one of the first results in set containment of inequality queries, giving a sufficient condition for it.

Theorem 4.2 *Let Q_1 and Q_2 be the following queries:*

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

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

where K_i's and F_i's are built-in predicates, i.e., inequalities.

Then, $Q_1 \leq_s Q_2$ if there is a containment mapping γ from Q_2 to Q_1 such that:

1. $\gamma(q(\vec{V})) = q(\vec{W})$*, that is, the head of Q_2 is mapped to the head of Q_1.*
2. $\forall i, 1 \leq i \leq k, \exists b, 1 \leq b \leq l$ *such that* $\gamma(p_i(\vec{Z_i})) = p_b(\vec{Y_b})$*, i.e., every ordinary subgoal of Q_2 is mapped to an ordinary subgoal of Q_1.*
3. *Every built-in predicate of Q_2, once γ is applied (i.e., every $\gamma(F_i)$) is implied by the built-in predicates of Q_1 (the K_i's).*

Proof:

The proof for this theorem is very similar to that of Theorem 4.1, but in this case the built-in predicates must be considered.

Assume there is such a containment mapping γ from Q_2 to Q_1. Consider an arbitrary ground database D from which Q_1 derives a fact t using an assignment mapping τ. That is, every $\tau(p_i(\vec{Y_i}))$ is a fact in D, and $t = \tau(q(\vec{W}))$.

Now, consider the consecutive application of γ and τ to Q_2. Applying them to the head of Q_2, we obtain $\tau(\gamma(q(\vec{V}))) = \tau(q(\vec{W})) = t$.

The application of $\tau \circ \gamma$ to every predicate in the body of Q_2 is always possible, since $\gamma(p_i(\vec{Z_i})) = p_j(\vec{Y_j})$, for some j ($1 \leq j \leq l$). Then, $\tau(\gamma(p_i(\vec{Z_i}))) = \tau(p_j(\vec{Y_j}))$, which is a fact in D. The built-in predicates of Q_2 will hold (every $\gamma(F_i)$), because they are implied by the built-in predicates of Q_1, which must be true in order to apply the assignment mapping that derives t. Therefore, every fact t derived by Q_1 using the assignment mapping τ is also derived by Q_2, using the assignment mapping $\tau \circ \gamma$. Thus, $Q_1 \leq_s Q_2$. □

Klug [Klu88] also sketches a proof for this theorem and shows that the existence of this containment mapping provides a necessary and sufficient condition for the set containment of a subclass of queries: left semiinterval queries and right semiinterval queries. *Left semiinterval queries* only admit inequalities of the form $X\theta c$, where X is a variable, c is a constant, and θ is one of $\leq$, $<$ or $=$. *Right semiinterval queries* only admit inequalities of the form $c\theta X$, being X, c and θ defined as above.

Besides, Klug gives a theorem [Klu88] that provides a necessary and sufficient condition to check whether, given two inequality queries Q_1 and Q_2 whose variables range over any *dense* and *totally ordered* domain, $Q_1 \leq_s Q_2$ holds. However, Klug stated in his paper that most of his results (including this theorem) do not hold for nondense domains like the integers.

Ron van der Meyden [vdM97] studied the problem of querying indefinite data over linear ordered domains. He demonstrated that one of the problems he dealt with was equivalent to the set containment of queries with inequalities, showing that it was decidable and Π_2^p-complete.

Using a different technique, based on counter machines, Ibarra and Su [IS97] show that the containment/equivalence problem is decidable for linear constraint queries (having an exponential time lower bound and an exponential space upper bound), but no effective procedure to test it is given.

In [BHPP98], we used canonical databases to sketch a necessary and sufficient condition and a procedure to test set containment of inequality queries. This was a preliminary work in the direction of the QCC procedure presented in this Thesis (Chapter 6).

4.4 Summary

This chapter showed the work that has been done about the set containment of equality and inequality conjunctive queries. Our contributions will be shown in chapters 5, for the set containment of equality queries, and 6, for the set containment of inequality queries.

Chapter 5

Applying QCC to test set containment of equality queries

Although this problem has already been solved by Chandra and Merlin [CM77], we show that QCC also works for this case. In fact, since an equality query is just an inequality query without any built-in predicate, the procedure shown later in Chapter 6 should also work for this particular 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 are built as shown in Algorithm 2 and do not need any further transformation. There is no need to add constraints, because the queries do not have built-in predicates, and the facts do not have symbolic multiplicities because the underlying semantics is set theoretic. Therefore, step (4) of Algorithm 2 can be omitted.

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. Then, we only try to find an assignment mapping from Q_2 to every canonical database to derive the canonical fact.

Step 3: Test the 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 the containment does not hold.

Example 5.1 Let Q_1 and Q_2 be the following equality queries:

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

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

It is obvious that Q_1 is set contained in Q_2, because Q_2 is more restrictive than Q_1. Using the results from [CM77], we can see that there is a containment mapping γ from Q_2 to Q_1 ($\gamma(U) = X$; $\gamma(V) = Y$). The presence of a containment mapping is the unique condition needed to prove the set containment of equality queries, therefore $Q_1 \leq_s Q_2$.

Set containment can also be proven by the use of QCC. The following table shows the two canonical databases in $CDBS(Q_1)$ and the assignment mappings from Q_2 that obtain the canonical facts from each canonical database. Q_2 obtains the canonical fact from all the canonical databases,

Table 5.1: Applying Q_2 to every canonical database

Name	Q-mapping	CDB	t_{d_i}	*Assignment mapping from* Q_2 *to obtain* t_{d_i}
	X Y	p	q	
d_1	A A	AA	A	$\tau_1(U) = \tau_1(V) = A$
d_2	A B	AB BA	A	$\tau_2(U) = A$; $\tau_2(V) = B$

using the assignment mappings shown in Table 5.1. Therefore, $Q_1 \leq_s Q_2$.
□

We leave the use of QCC to test set containment of equality queries without the proof of its correctness because it is a particular case of the set containment of inequality queries, shown in Chapter 6. However, the use of QCC for this case is very similar to the theorem offered by Chandra and Merlin [CM77]. Observe that the canonical database d_i built using a Q-mapping that maps every variable of Q_1 to a different uninterpreted constant (d_2 in the previous example) 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 to test the set containment of equality queries shown in [CM77]. Therefore, for this case of containment, there is no need to apply Q_2 to every $d_i \in CDBS(Q_1)$, but only to the canonical database built using the Q-mapping that maps every variable in the body of Q_1 to a different uninterpreted constant.

Chapter 6

Applying 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.

6.1 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 the 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. This 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). This has already been defined in Algorithm 2, before adapting canonical databases for each type of containment.

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 test the set containment, because it is not possible to build a ground database isomorphic to it from which Q_1 obtains the canonical fact.

Therefore, the step (4) of Algorithm 2 is the following.

$//Q - DBS = \{\theta_1(db(Q_1)), \ldots, \theta_{j-1}(db(Q_1))\}$

4. Generation of canonical databases
   For $i = 1$ to $j - 1$
   constraints$(d_i) = (\forall k, l \ (1 \leq k \neq l \leq q) \ \ A_k \neq A_l) \ \wedge$
   $(\theta_i(K_1) \wedge \cdots \wedge \theta_i(K_n))$
   if *constraints*(d_i) is satisfiable
   then {
   $d_i = \theta_i(db(Q_1))\}$
   }
   else $d_i = \emptyset$ //Discard d_i
   Return $CDBS(Q_1) = \{d_1, \ldots, d_{j-1}\}$ □

Example 6.1 Let Q_1 be the query

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

Table 6.1 shows the canonical database set for this query. The column $\theta_i(db(Q_1))$ shows the facts in the canonical database, and *constraints*(d_i) represents the constraints associated to this database. If *constraints*(d_i) is not satisfiable, the canonical database will not be considered.

Q_1 has 4 variables, therefore there are 15 possible nonisomorphic canonical databases for Q_1. However, not all of them are consistent with their constraints. The canonical databases d_2, d_3, d_6, d_9, and d_{15} are not consistent and will no longer be used, because it is not possible to build a ground database isomorphic to any of them. □

Table 6.1: (Example 6.1) $CDBS(Q_1)$

d_i	$\theta_i(db(Q_1))$		Q-Mapping	t_d	$constraints(d_i)$
	r	p	X Y Z U	q	
d_1	(A, D) (D, C)	(D, B)	A B C D	(A, B, C)	$A > B \wedge$ $A \neq B \wedge$ $A \neq C \wedge$ $A \neq D \wedge$ $B \neq C \wedge$ $B \neq D \wedge$ $C \neq D$
d_2	(A,B) (B,A)	(B,A)	A A A B	(A, A, A)	$A > A \wedge A \neq B$ $constraints(d_2)$ is unsatisfiable
d_3	(A,A) (A,B)	(A,A)	A A B A	(A, A, B)	$A > A \wedge A \neq B$ $constraints(d_3)$ is unsatisfiable
d_4	(A, A)	(A, B)	A B A A	(A, B, A)	$A > B \wedge A \neq B$
d_5	(B, A) (A, A)	(A, A)	B A A A	(B, A, A)	$B > A \wedge A \neq B$
d_6	(A,B) (B,B)	(B,A)	A A B B	(A, A, B)	$A > A \wedge A \neq B$ $constraints(d_6)$ is unsatisfiable
d_7	(A, B) (B, A)	(B, B)	A B A B	(A, B, A)	$A > B \wedge A \neq B$
d_8	(A, A) (A, B)	(A, B)	A B B A	(A, B, B)	$A > B \wedge A \neq B$
d_9	(A,C) (C,B)	(C,A)	A A B C	(A, A, B)	$A > A \wedge$ $A \neq B \wedge A \neq C \wedge B \neq C$ $constraints(d_9)$ is unsatisfiable
d_{10}	(A, C) (C, A)	(C, B)	A B A C	(A, B, A)	$A > B \wedge$ $A \neq B \wedge A \neq C \wedge B \neq C$
d_{11}	(A, A) (A, C)	(A, B)	A B C A	(A, B, C)	$A > B \wedge$ $A \neq B \wedge A \neq C \wedge B \neq C$
d_{12}	(B, C) (C, A)	(C, A)	B A A C	(B, A, A)	$B > A \wedge$ $A \neq B \wedge A \neq C \wedge B \neq C$
d_{13}	(B, A) (A, C)	(A, A)	B A C A	(B, A, C)	$B > A \wedge$ $A \neq B \wedge A \neq C \wedge B \neq C$
d_{14}	(B, A) (A, A)	(A, C)	B C A A	(B, C, A)	$B > C \wedge$ $A \neq B \wedge A \neq C \wedge B \neq C$
d_{15}	(A,A)	(A,A)	A A A A	(A, A, A)	$A > A$ $constraints(d_{15})$ is unsatisfiable

6.2 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 define how to apply Q_2

to a canonical database and how to know if Q_2 obtains the canonical fact. The following definition and lemma show how to do it.

Definition 6.1 $t_d \in Q_2(d)$:

Let d be a database in $CDBS(Q_1)$, and let t_d be the corresponding canonical fact. We say that $t_d \in Q_2(d)$ if for all ground substitutions α, defined on the variables in d, $\alpha(t_d) \in Q_2(\alpha(d))$.

Lemma 6.1 *Let Q_1 and Q_2 of the form*

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

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

where K_i's and F_i's are the built-in predicates. Let $d \in CDBS(Q_1)$.

Then $t_d \in Q_2(d)$ if and only if the following two conditions hold:

1. *There are assignment mappings $\tau_1, \ldots, \tau_s$ ($s \geq 1$) from the ordinary predicates of Q_2 to d such that $\tau_1(q(\vec{V})) = \cdots = \tau_s(q(\vec{V})) = t_d$, and*
2. *The formula $F = constraints(d) \wedge \neg(\tau_1(F_1) \wedge \cdots \wedge \tau_1(F_m)) \wedge \cdots \wedge \neg(\tau_s(F_1) \wedge \cdots \wedge \tau_s(F_m))$ is not satisfiable.*

Proof:

Before giving the formal proof, let us sketch it in an intuitive manner. The formula associated to each canonical database d has two parts: (1) $constraints(d)$, which is satisfiable (if it was unsatisfiable, the canonical database would not be considered); and (2) A conjunction of negated subformulas, each of which is the application of an assignment mapping (from Q_2 to d) to the built-in predicates in Q_2. All assignment mappings from Q_2 to d are included in this conjunction.

Given that $constraints(d)$ must hold, the unsatisfiability of the formula F indicates that the second part of the formula cannot be true. This means that at least one of the elements of this part is false. Each of these elements is the negation of the application of an assignment mapping τ_i from Q_2 to d, and without the negation it would be true. Therefore, there is always an assignment mapping from Q_2 to d that can be applied, so Q_2 obtains t_d.

ONLY IF**:** Condition 1 must be true, otherwise t_d cannot be in $Q_2(d)$. Now by contradiction we prove that condition 2 is also necessary to obtain $t_d \in Q_2(d)$.

Consider all the assignment mappings from Q_2 to d. Suppose that F is satisfiable. Then there is a ground substitution α for the variables of F that makes the formula F *true*. Since F is a conjunction of constraints, every individual constraint must be true in order for F to become true.

Focusing on the second part of the formula, for the mentioned ground substitution α, every element $\neg\alpha(\tau_j(F_1) \wedge \cdots \wedge \tau_j(F_m))$ is true. Therefore, for all j, $1 \leq j \leq s$, $\alpha(\tau_j(F_1) \wedge \cdots \wedge \tau_j(F_m))$ is $false$.

Using the constants in the substitution, we can define a ground database $\alpha(d)$ such that none of the assignment mappings from Q_2 can be applied, because the application of the assignment mapping to the built-in predicates of Q_2 does not hold. Thus, $\alpha(t_d) \notin Q_2(\alpha(d))$. and, by Definition 6.1, $t_d \notin Q_2(d)$. Therefore, F must be unsatisfiable in order to get $t_d \in Q_2(d)$.

If: Assume conditions 1 and 2 hold. By condition 2, F is not satisfiable. Since F is not satisfiable, and $constraints(d)$ must hold, there must be some element $\neg(\tau_i(F_1) \wedge \cdots \wedge \tau_i(F_m))$ $(1 \leq i \leq s)$ that is false for any ground substitution α.

That is, for all α there exists an i $(1 \leq i \leq s)$ such that $\alpha(\neg(\tau_i(F_1) \wedge \cdots \wedge \tau_i(F_m)))$ is false. Therefore, $\alpha(\tau_i(F_1) \wedge \cdots \wedge \tau_i(F_m))$ is true.

That means that, for any ground substitution α, there exists an assignment mapping τ_i from Q_2 to d that satisfies the built-in predicates in Q_2, so Q_2 obtains the canonical fact. Then, for any ground substitution α, $\alpha(t_d) \in Q_2(\alpha(d))$, which (by Definition 6.1) means that $t_d \in Q_2(d)$.

□

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

Example 6.2 Let Q_1 and Q_2 be the following queries:

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

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

One of the canonical databases d_i that we can build from Q_1 is the following, where A, B, C, and D denote uninterpreted constants:

CDB		t_{d_i}	$constraints(d_i)$
r	p	q	
A B	C D D C	A B	$(A \neq B) \wedge (A \neq C) \wedge (A \neq D) \wedge (B \neq C) \wedge$ $(B \neq D) \wedge (C \neq D) \wedge (A > B)$

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

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

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

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

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

There is a procedure in Section 6.5 to check the satisfiability of this kind of formulas. However, for this example, due to the simplicity of the formula, this check can be done directly. 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, $q(A, B) \in Q_2(d)$. □

6.3 Step 3: Test the 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:

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

Section 6.6 shows several examples that illustrate how to use QCC to test the set containment of inequality queries.

6.4 Validation of QCC for the set containment of inequality queries

The following lemma proves that the application of an assignment mapping from an inequality query Q_1 to any ground database D is isomorphic to some canonical database $d_i \in CDBS(Q_1)$. Then, the main theorem for this case demonstrates that $\forall d \in CDBS(Q_1),\ t_d \in Q_2(d)$ constitutes a necessary and sufficient condition to prove set containment of inequality queries.

Lemma 6.2 *Let Q_1 be an inequality query of the form $q(\vec{W})$:- $p_1(\vec{Y_1}), \ldots, p_l(\vec{Y_l}), K_1, \ldots, K_n$. Let τ be an assignment mapping from Q_1 to a database D. Let $sd = \{\tau(p_1(\vec{Y_1})), \cdots, \tau(p_l(\vec{Y_l}))\}$; i.e., sd is the subset of D where τ maps the ordinary predicates of Q_1. Then sd is isomorphic to a canonical database $d \in CDBS(Q_1)$:*

$$\exists d_i \in CDBS(Q_1) \mid d_i \textit{ is isomorphic to } sd$$

Proof:

The first part of the proof for this lemma is identical to that of Lemma 3.1. Given that the canonical databases represent all the patterns of equalities among uninterpreted constants (these patterns are built by construction of the canonical databases), sd is isomorphic to a canonical database $d_i \in CDBS(Q_1)$.

However, in order for the isomorphism to hold, the constants in the facts of sd must satisfy $constraints(d_i)$. But this is also true, because $constraints(d_i)$ always represents the application of an assignment mapping from Q_1 to a ground database D (if $constraints(d)$ were unsatisfiable, the canonical database would not be considered). □

Theorem 6.1 *Let Q_1 and Q_2 of the form*

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

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

Then, $Q_1 \leq_s Q_2 \Longleftrightarrow \forall d \in CDBS(Q_1)\ \ t_d \in Q_2(d)$.
Proof:

ONLY IF: By contradiction. Assume that there is d, $d \in CDBS(Q_1)$, such that $t_d \notin Q_2(d)$. Then, by Definition 6.1, there exists a ground substitution α such that $\alpha(t_d) \notin Q_2(\alpha(d))$. Since, by construction of each $d \in CDBS(Q_1)$, $t_d \in Q_1(d)$, then $\alpha(t_d) \in Q_1(\alpha(d))$. Therefore, $Q_1 \not\leq_s Q_2$.

IF: We assume that $\forall d \in CDBS(Q_1),\ t_d \in Q_2(d)$, and we want to prove that $Q_1 \leq_s Q_2$. Therefore, we need to prove that, if for any arbitrary database D and any arbitrary derived fact $t \in Q_1(D)$, then $t \in Q_2(D)$.

Assume that $t \in Q_1(D)$. Then, there exists an assignment mapping τ that maps the ordinary predicates of Q_1 to some facts in D, such that $\tau(q(\vec{W})) = t$.

Let $sd = \{\tau(p_1(\vec{Y_1})), \cdots, \tau(p_l(\vec{Y_l}))\}$, that is, sd is the subset of facts in D mapped by the ordinary predicates in the body of Q_1 through the assignment mapping τ. By Lemma 6.2, sd is isomorphic to a database d in $CDBS(Q_1)$. Let α be the mapping that shows that isomorphism. Then $\alpha(t_d) = t$ and $\alpha(d) = sd$. From hypothesis, $t_d \in Q_2(d)$. Then $\alpha(t_d) \in Q_2(\alpha(d))$, so, by Definition 6.1, $t \in Q_2(sd)$. Hence $t \in Q_2(D)$.

So for any database D and for any derived fact t in $Q_1(D)$, $Q_2(D)$ contains t. Therefore, $Q_1 \leq_s Q_2$.

□

Corollary 1 *Given two inequality queries Q_1 and Q_2, Q_1 is set contained into Q_2 ($Q_1 \leq_s Q_2$) if and only if $\forall d \in CDBS(Q_1),\ Q_1(d) \subseteq_s Q_2(d)$.*
Proof:

ONLY IF: If there is a $d \in CDBS(Q_1)$ such that $Q_1(d) \not\subseteq_s Q_2(d)$, we can build a ground database D to show a counterexample that shows that $Q_1 \not\leq_s Q_2$.

IF: By construction, $t_d \in Q_1(d)$; by hypothesis, given that $\forall d \in CDBS(Q_1),\ Q_1(d) \subseteq_s Q_2(d)$, we have that $t_d \in Q_2(d)$. Then, $\forall d \in CDBS(Q_1),\ t_d \in Q_2(d)$. Using the previous theorem, we conclude that $Q_1 \leq_s Q_2$.

6.5 Testing Satisfiability

In section 6.2 we showed that, in order to know whether the canonical fact t_d is obtained when a query Q_2 is applied to a canonical database d, that is, to test if $t_d \in Q_2(d)$, it is necessary to test the satisfiability of a formula F.

There has been a lot of work about testing the satisfiability of a formula that is a conjunction of inequalities. Guo et al. [GSW96] offer excellent results in this field. However, when a nondense domain (such as the integers) is assumed, they only consider the operators $<$ and $\leq$; the $\neq$ operator is used only with dense domains such as the real numbers. There are also results about the satisfiability of a formula when the domain is dense in [ZO93, IO97].

We will offer here a procedure to test the satisfiability of a formula when the variables take their values from any ordered domain, either dense or nondense. However, our procedure is specific for the kind of formulas we are dealing with (the formula F shown in Lemma 6.1). These formulas will always have the form

$$F = constraints(d) \wedge \neg(\tau_1(F_1) \wedge \cdots \wedge \tau_1(F_m)) \wedge \cdots \wedge \neg(\tau_l(F_1) \wedge \cdots \wedge \tau_l(F_m))$$

where

$$constraints(d) = C_1 \wedge C_2 \wedge \cdots \wedge C_n \wedge \theta_d(K_1) \wedge \theta_d(K_2) \wedge \cdots \wedge \theta_d(K_r).$$

C_i's are the constraints that specify that all uninterpreted constants are different, θ_d is the Q-mapping used to build d, and K_i's are the built-in predicates of Q_1. $\tau_j(F_i)$ represents the built-in predicate F_i of Q_2 mapped to the uninterpreted constants in d by the assignment mapping τ_j.

C_i's are of the form $A_i \neq A_j$ where A_i and A_j are uninterpreted constants in A_Q, and both $\theta_d(K_i)$'s and $\tau_j(F_i)$'s are of the form $X < Y$, $X \leq Y$, $X = Y$ or $X \neq Y$, where any of X or Y (but not both) can be a constant of the domain, or X and Y can be two uninterpreted constants coming from the facts in the canonical database.

The procedure to test the satisfiability of the formula F is the following.

1. **Normalize the formula**. It involves three steps: eliminate the negations of conjunctions of constraints; simplify the formula; and check for equalities among uninterpreted constants. The output of this step will be the normalized formula or the early decision that it is not satisfiable.

   (a) **Eliminate the negations of conjunctions of constraints**.
   First, we apply DeMorgan's Law to the negations of conjunctions of constraints, obtaining

   $$F = C_1 \wedge C_2 \wedge \cdots C_n \;\wedge\; \theta_d(K_1) \wedge \theta_d(K_2) \wedge \cdots \wedge \theta_d(K_r) \;\wedge$$

   $$(\neg\tau_1(F_1) \vee \cdots \vee \neg\tau_1(F_m)) \;\wedge\; \cdots \;\wedge\; (\neg\tau_l(F_1) \vee \cdots \vee \neg\tau_l(F_m))$$

   Second, since $\tau_j(F_i)$'s are of the form $X < Y$, $X \leq Y$, $X = Y$ or $X \neq Y$, we remove the negations of atomic constraints by applying the following equivalences: $\neg(A < B) \equiv A \geq B$, $\neg(A \leq B) \equiv A > B$, $\neg(A = B) \equiv A \neq B$, and $\neg(A \neq B) \equiv A = B$.

Denoting L_{ij} the constraint equivalent to $\neg\tau_i(F_j)$, the formula F can be rewritten as

$$C_1 \ \wedge\ C_2 \ \wedge\ \cdots C_n \ \wedge\ \theta_d(K_1) \ \wedge\ \theta_d(K_2) \ \wedge\ \cdots \ \wedge\ \theta_d(K_r) \ \wedge$$

$$(L_{11} \vee L_{12} \vee \cdots \vee L_{1m}) \ \wedge\ \cdots \ \wedge\ (L_{l1} \vee L_{l2} \vee \cdots \vee L_{lm})$$

Finally, we apply the distributive law of $\wedge$ with respect to $\vee$, obtaining

$$[C_1 \wedge \cdots \wedge C_n \wedge \theta_d(K_1) \wedge \theta_d(K_2) \wedge \cdots \wedge \theta_d(K_r) \wedge L_{11} \wedge \cdots \wedge L_{l1}]$$

$$\vee \cdots \vee$$

$$[C_1 \wedge \cdots \wedge C_n \wedge \theta_d(K_1) \wedge \theta_d(K_2) \wedge \cdots \wedge \theta_d(K_r) \wedge L_{1m} \wedge \cdots \wedge L_{lm}]$$

Let us denote each element of the disjunction (shown in the above formula enclosed in square brackets) by P_i. Then, $F = P_1 \vee P_2 \vee \cdots \vee P_w$.

(b) **Simplify the formula**.

A special characteristic of the formula F is that *all* its uninterpreted constants are different (these constraints come from $constraints(d)$). Therefore, for each $A_i \leq A_j$ we have a constraint $A_i \neq A_j$, and we can replace $A_i \leq A_j$ by $A_i < A_j$.

The second simplification that can be applied to the formula is to remove duplicates of constraints. For example, if there is a conjunction such as $(\cdots \ \wedge\ (A > B) \ \wedge\ \cdots \ \wedge\ (A > B) \ \wedge\ \cdots)$, we can remove all the duplicates of the $(A > B)$ atomic constraints, leaving just one.

(c) **Check for equalities among uninterpreted constants**.

If there is a subformula P_k with a constraint $A_i = A_j$, this P_k will have also a constraint $A_i \neq A_j$ because each P_k has all the constraints C_k. Therefore, the part of the formula F expressed by this P_k is clearly not satisfiable and we remove this P_k from F. If all the P_k's in F are removed and F becomes empty, then we can conclude that F is not satisfiable and this procedure stops here.

2. **Build a directed graph** $G(P_i)$ **for each** P_i. The nodes of the graph are the variables in P_i. The arcs of the graph are added as follows:

   - If there is a constraint $X < Y$ in P_i, we will draw a solid arc from X to Y.

- If there is a constraint $X \leq Y$ in P_i, we will draw a dashed arc from X to Y. Note that, for such a constraint to exist, either X or Y is a constant, otherwise it would be converted into a $<$ constraint in step (1b).
- If there is a constraint $X = Y$, we will draw a dashed arc from X to Y and another one from Y to X (As if replacing $X = Y$ by $X \leq Y \ \wedge \ Y \leq X$). As in the previous case, either X or Y must be a constant.
- Each pair of (different) constants a and b will be connected by a solid arc, from a to b if $a < b$, or from b to a if $b < a$.

3. **Check the satisfiability of each graph** $G(P_i)$.

   It is necessary to consider two different situations: when variables[1] range over a dense domain such as the real numbers, and when variables range over a nondense domain such as the integers. In the first case it is only necessary to check whether the first of the two following conditions is satisfied. But in the second case, when variables range over a nondense domain, the two next conditions must be satisfied.

   Condition 1: If there is a cycle with at least one solid arc in the graph $G(P_i)$, the subformula P_i is unsatisfiable (that would mean that a variable is strictly less than itself).

   Condition 2: The idea in this condition is to check whether there is "enough room" for all the variables that are placed between two constants. This is only a problem if there are two or more constants in the graph connected by one or more paths, and the variables range over integers (or other nondense domain).

   Let a and b be two constants in the graph connected through one or more paths. Note that in these paths there can be a dashed arc only between a variable and a constant, but never between two variables.

   Let s be the set of symbols that includes a, b and all constants and variables in all paths between a and b, that is

   $$s = \{S | S = a \text{ or } S = b \text{ or } S \text{ is a symbol in any path from } a \text{ to } b\}.$$

   Let us call I the ordering of the set of consecutive integers between a and b, both of them included, that is,

   $$I = \langle a, a+1, \ldots b-1, b \rangle$$

[1] It is possible to consider our uninterpreted constants as variables in this context

We are looking for a mapping β from symbols in s to the ordering I. Such an assignment mapping must satisfy

(a) $\beta(a) = a$.

(b) $\beta(b) = b$.

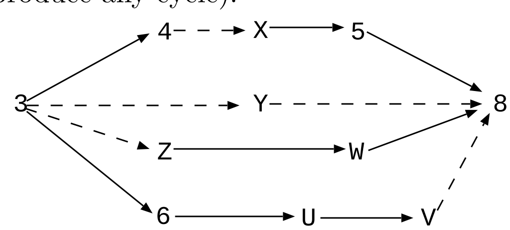
(c) For all $c \in s$, if c is a constant, then $\beta(c) = c$, that is, β is the identity for constants[2].

(d) For all variables $X, Y \in s$, $X \neq Y$ implies $\beta(X) \neq \beta(Y)$.

(e) The relationships expressed in the subformula P_i (from which the graph $G(P_i)$ was built) between two symbols S_i and $S_j \in s$ must be preserved by the mapping.

If a graph $G(P_i)$ satisfies Condition 1 and for every pair of constants a and b with at least one path between them in the graph $G(P_i)$, such an assignment mapping β is found, the formula P_i is satisfiable, otherwise it is not. Notice that if such assignment mapping exists, it will always be found.

Example 6.3 Let P_i be represented by the following graph (the solid arcs between some pairs of constants are omitted for clarity; they would not produce any cycle):



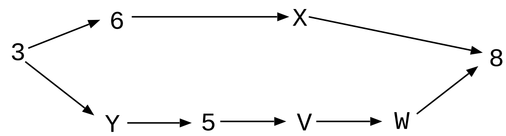
Let us consider the constants 3 and 8. For this example, $s = \{3, 4, X, 5, Y, Z, W, 6, U, V, 8\}$ and $I = \langle 3, 4, 5, 6, 7, 8 \rangle$. The next table shows one of the possible assignment mappings β

$\beta(3) = 3$	$\beta(4) = 4$	$\beta(X) = 4$
$\beta(5) = 5$	$\beta(Y) = 3$	$\beta(Z) = 5$
$\beta(W) = 6$	$\beta(6) = 6$	$\beta(U) = 7$
$\beta(V) = 8$	$\beta(8) = 8$	

Notice that each variable is mapped to a different constant and that all the relationships expressed by the formula are preserved. The graph has no cycles (condition 1) and the assignment mapping β was found, therefore P_i is satisfiable. □

[2] Evidently, this condition implies the two previous ones.

Example 6.4 Let P_i be represented by the following graph, and consider again the constants 3 and 8.



In this case, $s = \{3, 6, X, Y, 5, V, W, 8\}$ and $I = \langle 3, 4, 5, 6, 7, 8 \rangle$. P_i satisfies condition 1, because the graph has no cycles. But it is not possible to find an assignment mapping such that each variable is mapped to a different constant preserving the relationships between them. Therefore, P_i is not satisfiable. □

4. **Output the result**. Recall that the formula F was transformed into a disjunction of subformulas:

$$F = P_1 \vee P_2 \vee \cdots \vee P_w.$$

If all P_i's are unsatisfiable, then the formula F is unsatisfiable. If there exists a P_i that is satisfiable, the formula F is satisfiable.

Let us see an example:

Example 6.5 The formula F:

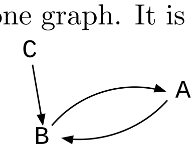
$$F = A \neq B \quad \wedge \quad B \neq C \quad \wedge \quad A \neq C \quad \wedge \quad B \leq A \quad \wedge \quad B < C \quad \wedge \quad \neg(B < A)$$

is not satisfiable. Let us use the algorithm to prove it:

1. The normalized formula (without negated conjunctions of constraints and with the $\leq$ converted into $<$) is

$$A \neq B \;\wedge\; B \neq C \wedge A \neq C \wedge B < A \wedge B < C \wedge A < B$$

2. We have to build only one graph. It is shown in the following picture.



3. The formula is not satisfiable, because the (only) P_i is not satisfiable: there is a cycle with at least one solid line in the graph.

□

6.5.1 Implications of the domain of the variables

The underlying domain of the variables has important implications in the problem of containment of inequality queries. The results obtained by Klug [Klu88] apply only when the domain is dense, like the real numbers; they do not apply for nondense domain such as the integers.

The IC-RFT (Implication Constraint Refutation) problem, shown to be polynomially equivalent to the query containment problem by Klug [Klu88] was studied in [ZO93]. However, when the domain was nondense, only the $=$ and $\neq$ operators were considered. Guo et al. [GSW96] solved the satisfiability of a conjunction of constraints when the domain was dense; when it was nondense, they did not allow the $\neq$ operator in the formulas.

The difficulty added by nondense versus dense domains is that the test for satisfiability must check if there is "enough room" for all the variables that must fit between two constants. For example, the formula $(X \neq Y) \wedge (3 < X) \wedge (X < Y) \wedge (Y < 5)$ is satisfiable if the domain is the real numbers (nondense) (with X=3.5 and Y=3.6, for example), while it is not satisfiable for the integers: there is not enough space to fit two integer values between 3 and 5.

With the second condition we gave in our procedure to test the satisfiability of a formula, this problem is also considered. Therefore, our procedure is valid for dense as well as nondense domains.

Given that the unsatisfiability of the formula F introduced in Lemma 6.1 is the only test (applied to all canonical databases) needed to test the set containment of inequality queries, the differences between the underlying domains lead us to some interesting conclusions. Let us use $\mathbb{R}$, the real numbers, as a representative of dense domains, and $\mathbb{Z}$, the integers, as a representative of nondense domains. We shall denote query containment under $\mathbb{R}$ as $\leq_s^{\mathbb{R}}$, and query containment under $\mathbb{Z}$ as $\leq_s^{\mathbb{Z}}$.

- If F is unsatisfiable in $\mathbb{R}$, then F is unsatisfiable in $\mathbb{Z}$. That means that if a query Q_1 is set contained in another query Q_2 under $\mathbb{R}$, it is also set contained under $\mathbb{Z}$: $Q_1 \leq_s^{\mathbb{R}} Q_2 \Longrightarrow Q_1 \leq_s^{\mathbb{Z}} Q_2$.

- if F is unsatisfiable in $\mathbb{Z}$, F may be satisfiable or unsatisfiable in $\mathbb{R}$. Therefore, $Q_1 \leq_s^{\mathbb{Z}} Q_2 \not\Longrightarrow Q_1 \leq_s^{\mathbb{R}} Q_2$.

- if F is satisfiable in $\mathbb{Z}$, then F is satisfiable in $\mathbb{R}$. Therefore, $Q_1 \not\leq_s^{\mathbb{Z}} Q_2 \Longrightarrow Q_1 \not\leq_s^{\mathbb{R}} Q_2$.

6.6 Examples

In this section we will show three examples. The first one shows two inequality queries that satisfy the containment (i.e., $Q_1 \leq_s Q_2$). For the second example, Q_1 is not contained in Q_2, and we shall use a canonical database to show a counterexample. The last example shows two queries for which the containment depends upon the underlying domain.

Example 6.6 Let Q_1 and Q_2 be the following queries:

$$Q_1 \;:\; q(X,Y) \text{ :- } p(X,Y), p(Y,Z), s(X,T), s(T,X), X \geq Y, Y > Z.$$

$$Q_2 \;:\; q(X,Y) \text{ :- } p(X,Y), p(X,V), s(M,W), X \geq V, M \geq W.$$

Using the results from Ullman [Ull89] for these two queries, it is not possible to check if $Q_1 \leq_s Q_2$:

There are two containment mappings τ_1 and τ_2 from Q_2 to Q_1:

$$\gamma_1(X) = X; \gamma_1(Y) = Y; \gamma_1(V) = Y; \gamma_1(M) = X; \gamma_1(W) = T$$

$$\gamma_2(X) = X; \gamma_2(Y) = Y; \gamma_2(V) = Y; \gamma_2(M) = T; \gamma_2(W) = X$$

But under none of these mappings can we imply the built-in predicates of Q_2. That is, we cannot imply either of $\gamma_1(X \geq V \;\wedge\; M \geq W)$ or $\gamma_2(X \geq V \;\wedge\; M \geq W)$ from the built-in predicates of Q_1, $X \geq Y \;\wedge\; Y > Z$ (more specifically, we cannot imply either $\gamma_1(M \geq W) = X \geq T$ or $\gamma_2(M \geq W) = T \geq X$).

Considering a nondense domain like the integers for the variables, the results of Klug [Klu88] are not useful either. However, applying our procedure, which is summarized in Table 6.2, we can conclude that $Q_1 \leq_s Q_2$.

Let us follow the steps of QCC for this case:

Step 1: Build $CDBS(Q_1)$

Table 6.2 shows (in the column labeled $CDBS(Q_1)$) the canonical databases built from the body of Q_1. From the 15 canonical databases that are possible, there are only 7 that are consistent with their constraints: d_3, d_6, d_9, d_{11}, d_{13}, d_{14} and d_{15}. The procedure will need to deal with only these databases to check the containment.

Step 2: Apply Q_1 and Q_2 to the canonical databases

The third column in the table shows t_d, the canonical fact that Q_2 should obtain, the mappings we can use to get it, and the formula F that is built as seen in Lemma 6.1.

Table 6.2: Example of containment

$Q-mappings$				$CDBS(Q_1)$			t_d	Mappings from Q_2					$Formula$
X	Y	Z	T	d_i	p	s	q	X	Y	V	M	W	
A	A	A	A	d_1	AA	AA							
				$A \geq A \wedge A > A$			d_1 is not consistent						
				d_2	AA	AB							
A	A	A	B			BA	d_2 is not consistent						
				$A \neq B \wedge A \geq A \wedge A > A$									
				d_3	AA	AA							$A \neq B \wedge A \geq A \wedge A > B \wedge$
A	A	B	A		AB		AA	A	A	A	A	A	$\neg(A \geq A \wedge A \geq A) \wedge$
				$A \neq B \wedge A \geq A \wedge A > B$				A	A	B	A	A	$\neg(A \geq B \wedge A \geq A)$
				d_4	AB	AA							
A	B	A	A		BA		d_4 is not consistent						
				$A \neq B \wedge A \geq B \wedge B > A$									
				d_5	BA	BA							
B	A	A	A		AA	AB	d_5 is not consistent						
				$A \neq B \wedge B \geq A \wedge A > A$									
				d_6	AA	AB							$A \neq B \wedge A \geq A \wedge A > B \wedge$
A	A	B	B		AB	BA	AA	A	A	A	A	B	$\neg(A \geq A \wedge A \geq B) \wedge$
				$A \neq B \wedge A \geq A \wedge A > B$				A	A	A	B	A	$\neg(A \geq A \wedge B \geq A) \wedge$
								A	A	B	A	B	$\neg(A \geq B \wedge A \geq B) \wedge$
								A	A	B	B	A	$\neg(A \geq B \wedge B \geq A)$
				d_7	AB	BA							
A	B	A	B		BA	BA	d_7 is not consistent						
				$A \neq B \wedge A \geq B \wedge B > A$									
				d_8	AB	AA							
A	B	B	A		BB		d_8 is not consistent						
				$A \neq B \wedge A \geq B \wedge B > B$									
				d_9	AA	AC							$A \neq B \wedge A \neq C \wedge B \neq C \wedge$
A	A	B	C		AB	CA	AA	A	A	A	A	C	$A \geq A \wedge A > B \wedge$
				$A \neq B \wedge A \neq C \wedge B \neq C \wedge$				A	A	A	C	A	$\neg(A \geq A \wedge A \geq C) \wedge$
				$A \geq A \wedge A > B$				A	A	B	A	C	$\neg(A \geq A \wedge C \geq A) \wedge$
								A	A	B	C	A	$\neg(A \geq B \wedge A \geq C) \wedge$
													$\neg(A \geq B \wedge C \geq A)$
				d_{10}	AB	AC							
A	B	A	C		BA	CA	d_{10} is not consistent						
				$A \neq B \wedge A \neq C \wedge B \neq C \wedge$									
				$A \geq B \wedge B > A$									
				d_{11}	AB	AA							$A \neq B \wedge A \neq C \wedge B \neq C \wedge$
A	B	C	A		BC		AB	A	B	B	A	A	$A \geq B \wedge B > C \wedge$
				$A \neq B \wedge A \neq C \wedge B \neq C \wedge$									$\neg(A \geq B \wedge A \geq A)$
				$A \geq B \wedge B > C$									
				d_{12}	BA	BC							
B	A	A	C		AA	CB	d_{12} is not consistent						
				$A \neq B \wedge A \neq C \wedge B \neq C \wedge$									
				$B \geq A \wedge A > A$									
				d_{13}	BA	BA							$A \neq B \wedge A \neq C \wedge B \neq C \wedge$
B	A	C	A		AC	AB	BA	B	A	A	B	A	$B \geq A \wedge A > C \wedge$
				$A \neq B \wedge A \neq C \wedge B \neq C \wedge$				B	A	A	A	B	$\neg(B \geq A \wedge B \geq A) \wedge$
				$B \geq A \wedge A > C$									$\neg(B \geq A \wedge A \geq B)$
				d_{14}	BC	BA							$A \neq B \wedge A \neq C \wedge B \neq C \wedge$
B	C	A	A		CA	AB	BC	B	C	C	B	A	$B \geq C \wedge C > A \wedge$
				$A \neq B \wedge A \neq C \wedge B \neq C \wedge$				B	C	C	A	B	$\neg(B \geq C \wedge B \geq A) \wedge$
				$B \geq C \wedge C > A$									$\neg(B \geq C \wedge A \geq B)$
				d_{15}	AB	AD							$A \neq B \wedge A \neq C \wedge B \neq C \wedge$
A	B	C	D		BC	DA	AB	A	B	B	A	D	$A \geq B \wedge B > C \wedge$
				$A \neq B \wedge A \neq C \wedge B \neq C \wedge$				A	B	B	D	A	$\neg(A \geq B \wedge A \geq D) \wedge$

In order to verify that $t_d \in Q_2(d)$ we only need to check the unsatisfiability of the formula F for every canonical database, using the algorithm shown in Section 6.5. Let us follow the algorithm to check that the formula corresponding to d_3 is unsatisfiable.

1. The original formula is

$$A \neq B \ \wedge\ A \geq A \ \wedge\ A > B \ \wedge\ \neg(A \geq A \ \wedge\ A \geq A) \\ \wedge\ \neg(A \geq B \ \wedge\ A \geq A)$$

2. Normalize the formula:

$$\begin{gathered} A \neq B \ \wedge\ A \geq A \ \wedge\ A > B \ \wedge\ \neg(A \geq A) \ \wedge\ \neg(A \geq B \ \wedge\ A \geq A) \\ = \\ A \neq B \ \wedge\ A \geq A \ \wedge\ A > B \ \wedge\ (A < A) \ \wedge\ (A < B \vee A < A) \\ = \\ [A \neq B \ \wedge\ A \geq A \ \wedge\ A > B \ \wedge\ A < A \ \wedge\ A < B)] \ \vee \\ [A \neq B \ \wedge\ A \geq A \ \wedge\ A > B \ \wedge\ A < A \ \wedge\ A < A] \end{gathered}$$

3. Build a directed graph for each subformula.

   We should build two directed graphs, but just looking at the formula we find the term $(A < A)$ in both subformulas. Thus, the formula is unsatisfiable, and we have that $t_{d_2} \in Q_2(d_2)$.

Step 3: Test the containment

It is easy to follow the same algorithm for the rest of the canonical databases, checking that all of their corresponding formulas are unsatisfiable. So, for all consistent canonical databases, we have $t_{d_i} \in Q_2(d_i)$, and we can conclude that $Q_1 \leq_s Q_2$.

□

Example 6.7 Let us slightly modify the queries from the previous example, and be Q_1 and Q_2 the following:

$$Q_1 \ :\ q(X,Y) \text{ :- } p(X,Y), p(Y,Z), s(X,T), s(T,X), X \geq Y, Y > Z.$$

$$Q_2 \ :\ q(X,Y) \text{ :- } p(X,Y), p(X,V), s(X,W), X \geq V, X \leq W.$$

Let us focus in just one canonical database, d_6. Since Q_1 is the same as the previous example, all the canonical databases are identical as the ones shown in Table 6.2. The difference in the result comes from applying Q_2 (which was the modified query) to the databases.

We can see that, applying Q_2 to d_6, the resulting formula is satisfiable:

Table 6.3: Example of non containment

$Q-mappings$				$CDBS(Q_1)$			t_d	$Mappings$				$Formula$
X	Y	Z	T	d_i	p	s	q	X	Y	V	W	
A	A	B	B	d_6	AA AB	AB BA	AA	A A	A A	A B	B B	$A \neq B \wedge A \geq A \wedge A > B \wedge$ $\neg(A \geq A \wedge A \leq B) \wedge$ $\neg(A \geq B \wedge A \leq B)$
				$A \neq B \wedge A \geq A \wedge A > B$								

$$A \neq B \wedge A \geq A \wedge A > B \wedge \neg(A \geq A \wedge A \leq B) \wedge \neg(A \geq B \wedge A \leq B)$$

$$=$$

$$A \neq B \wedge A \geq A \wedge A > B \wedge (A < A \vee A > B) \wedge (A < B \vee A > B)$$

$$=$$

$$[A \neq B \wedge A \geq A \wedge A > B \wedge \mathbf{A} < \mathbf{A} \wedge A < B] \text{ (unsatisfiable)}$$
$$\vee$$
$$[A \neq B \wedge A \geq A \wedge A > B \wedge \mathbf{A} < \mathbf{A} \wedge A > B] \text{ (unsatisfiable)}$$
$$\vee$$
$$[A \neq B \wedge A \geq A \wedge A > B \wedge \mathbf{A} > \mathbf{B} \wedge \mathbf{A} < \mathbf{B}] \text{ (unsatisfiable)}$$
$$\vee$$
$$[A \neq B \wedge A \geq A \wedge A > B \wedge A > B \wedge A > B] \text{ (satisfiable)}$$

As it can be seen, this formula is satisfiable, so $t_{d_6} = q(A, A) \notin Q_2(d_6)$

We can build a ground database isomorphic to d_6, using for example integers, and verify that the containment does not hold:

p		s	
3	3	3	2
3	2	2	2

Over this database, Q_1 obtains the tuple $q(3, 3)$, while Q_2 does not. Therefore, $Q_1 \not\leq_s Q_2$. □

Example 6.8 Let Q_1 and Q_2 be the following inequality queries:

$Q_1 :\quad q(X, Y)$:- $p(X, Y), r(X, W), r(W, X), X > Y, Y > 3, W > 4.$

$Q_2 :\quad q(X, Y)$:- $p(X, Y), r(M, N), M \geq N, N > 4.$

For these queries, $constraints(d_1)$ and $constraints(d_2)$ are not satisfiable, therefore d_1 and d_2 are not taken into account to test query containment. We must check whether the formulas generated for d_3, d_4 and d_5 are not satisfiable.

Table 6.4: Example domain-dependent containment

Q-mappings			$CDBS(Q_1)$			t_d	*Mappings*				Formula
X	Y	W	d_i	p	r	q	X	Y	M	N	
A	A	A	d_1	AA	AA	AA					d_1 *is not consistent*
			$A > A \wedge A > 3 \wedge A > 4$								
A	A	B	d_2	AA	AB BA	AA					d_2 *is not consistent*
			$A \neq B \wedge A > A \wedge$ $A > 3 \wedge B > 4$								
A	B	A	d_3	AB	AA	AB	A	B	A	A	$A \neq B \wedge A > B \wedge B > 3 \wedge A > 4 \wedge$ $\neg(A \geq A \wedge A > 4)$
			$A \neq B \wedge A > B \wedge$ $B > 3 \wedge A > 4$								
B	A	A	d_4	BA	BA AB	BA	B B	A A	B A	A B	$A \neq B \wedge B > A \wedge A > 3 \wedge A > 4 \wedge$ $\neg(B \geq A \wedge A > 4 \wedge)$ $\neg(A \geq B \wedge B > 4)$
			$A \neq B \wedge B > A \wedge$ $A > 3 \wedge A > 4$								
A	B	C	d_5	AB	AC CA	AB	A A	B B	A C	C A	$A \neq B \wedge A \neq C \wedge B \neq C \wedge$ $A > B \wedge B > 3 \wedge C > 4 \wedge$ $\neg(A \geq C \wedge C > 4) \wedge$ $\neg(C \geq A \wedge A > 4)$
			$A \neq B \wedge A \neq C \wedge B \neq C \wedge$ $A > B \wedge B > 3 \wedge C > 4$								

d_3**:** The formula F for d_3 is

$$A \neq B \wedge A > B \wedge B > 3 \wedge A > 4 \wedge \neg(A \geq A \wedge A > 4)$$

$$\equiv$$

$$A \neq B \wedge A > B \wedge B > 3 \wedge A > 4 \wedge (A < A \vee A \leq 4)$$

$$\equiv$$

$$[A \neq B \wedge A > B \wedge B > 3 \wedge A > 4 \wedge \mathbf{A} < \mathbf{A}] \vee$$
$$[A \neq B \wedge A > B \wedge B > 3 \wedge \mathbf{A} > \mathbf{4} \wedge \mathbf{A} \leq \mathbf{4}]$$

It can be easily checked, without using the graph, that the two subformulas are unsatisfiable (the inequalities that make them unsatisfiable are shown in boldface). Therefore, the formula F is unsatisfiable.

d_4**:** The formula F for this canonical database is also unsatisfiable, since all its subformulas are unsatisfiable.

$$A \neq B \wedge B > A \wedge A > 3 \wedge A > 4 \wedge \neg(B \geq A \wedge A > 4) \wedge \neg(A \geq B \wedge B > 4)$$

$$\equiv$$

$$A \neq B \wedge B > A \wedge A > 3 \wedge A > 4 \wedge (B < A \vee A \leq 4) \wedge (A < B \vee B \leq 4)$$

$$\equiv$$

$$[A \neq B \ \wedge \ B > A \ \wedge \ A > 3 \ \wedge \ A > 4 \ \wedge \ \mathbf{B} < \mathbf{A} \ \wedge \ \mathbf{A} < \mathbf{B}] \ \vee$$

$$[A \neq B \ \wedge \ \mathbf{B} > \mathbf{A} \ \wedge \ A > 3 \ \wedge \ A > 4 \ \wedge \ \mathbf{B} < \mathbf{A} \ \wedge \ B \leq 4] \ \vee$$

$$[A \neq B \ \wedge \ B > A \ \wedge \ A > 3 \ \wedge \ \mathbf{A} > \mathbf{4} \ \wedge \ \mathbf{A} \leq \mathbf{4} \ \wedge \ A < B] \ \vee$$

$$[A \neq B \ \wedge \ B > A \ \wedge \ A > 3 \ \wedge \ \mathbf{A} > \mathbf{4} \ \wedge \ \mathbf{A} \leq \mathbf{4} \ \wedge \ B \leq 4]$$

d_5**:** The formula F for this canonical database is:

$$A \neq B \ \wedge \ A \neq C \ \wedge \ B \neq C \ \wedge \ A > B \ \wedge \ B > 3 \ \wedge \ C > 4 \ \wedge$$

$$\neg(A \geq C \ \wedge \ C > 4) \ \wedge \ \neg(C \geq A \ \wedge \ A > 4)$$

$$\equiv$$

$$A \neq B \ \wedge \ A \neq C \ \wedge \ B \neq C \ \wedge \ A > B \ \wedge \ B > 3 \ \wedge \ C > 4 \ \wedge$$

$$(A < C \ \vee \ C \leq 4) \ \wedge \ (C < A \ \vee \ A \leq 4)$$

$$\equiv$$

$$[A \neq B \wedge A \neq C \wedge B \neq C \wedge A > B \wedge B > 3 \wedge C > 4 \wedge \mathbf{A} < \mathbf{C} \ \wedge \ \mathbf{C} < \mathbf{A}] \vee$$

$$[A \neq B \wedge A \neq C \wedge B \neq C \wedge A > B \wedge B > 3 \wedge C > 4 \wedge A < C \wedge A \leq 4] \vee$$

$$[A \neq B \wedge A \neq C \wedge B \neq C \wedge A > B \wedge B > 3 \wedge \mathbf{C} > \mathbf{4} \ \wedge \ \mathbf{C} \leq \mathbf{4} \wedge C < A] \vee$$

$$[A \neq B \wedge A \neq C \wedge B \neq C \wedge A > B \wedge B > 3 \wedge \mathbf{C} > \mathbf{4} \ \wedge \ \mathbf{C} \leq \mathbf{4} \wedge A \leq 4]$$

For this formula, the first, third and fourth subformulas are clearly unsatisfiable. Let us build the graph for the second subformula, shown in Figure 6.1.

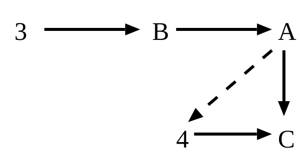


Figure 6.1: Domain-dependent satisfiability of a formula.

There are no cycles with a solid arc in the graph (in fact, there are no cycles at all). This is the only condition needed to test the satisfiability of the formula with a dense domain, thus the formula is satisfiable for dense domains such as the real numbers (for example, mapping A, B and C to

3.1, 3.2 y 3.3, the subformula becomes true, and so does the formula F). Therefore, using Theorem 6.1, we can conclude that $Q_1 \not\leq_s Q_2$ when the domain is dense.

However, when the underlying domain is nondense, such as the integers, the results are different. There are two conditions that must be checked to test the satisfiability of the formula:

1. The corresponding graph has no cycles with at least a solid arc. As we have seen in the previous graph, the graph has no cycles.

2. It must be checked if there is "enough room" for all the variables and constants that can be found between two constants connected by at least one path in the graph. In this case, there is a path between the constants 3 and 4. Using our algorithm, we must find a mapping from s, the set of variables and constants in the graph, to I, the list of consecutive integers between 3 and 4. For this graph, we have $s = \{3, B, A, C, 4\}$, and $I = \langle 3, 4 \rangle$. The constraints in the graph, $3 < B \wedge B < A \wedge A \leq 4$, must hold, too. It is easy to check that such mapping does not exist, so the formula is not satisfiable.

This second condition does not hold, therefore this subformula is unsatisfiable. We had already shown that the remaining subformulas (independently from the domain) were always unsatisfiable, thus F is not satisfiable. Therefore, for nondense domains, $Q_1 \leq_s Q_2$. □

6.7 Summary

This chapter has presented another original contribution of this Thesis. We have shown how *QCC* can be applied to test the set containment of inequality queries, by adding some constraints to each canonical database. The problem is then reduced to the test of the unsatisfiability of a formula composed of equalities and inequalities, and we have presented a procedure to perform this test.

The application of *QCC* to test the set containment of inequality queries finally solves an open problem since [Klu88], where Klug left it open for the cases when variables took their values from nondense domains.

This chapter closes the part of the thesis dedicated to the problem of the set containment of conjunctive queries. The following chapters will show the previous work and our contributions for the problem of bag containment of conjunctive queries.

Chapter 7

Previous work about bag containment of conjunctive queries

7.1 Introduction

The study of the containment of conjunctive queries under bag semantics has not been as extensive as under set semantics. Besides, the results achieved for the set containment problem of conjunctive query containment do not apply to the bag containment problem, because set containment does not imply bag containment, as shown in Example 1.1.

However, Brisaboa and Hernández [BH97] gave a necessary and sufficient condition, along with a procedure, to test bag containment of equality queries.

There has been no results, to the best of our knowledge, that offer a condition to test bag containment of inequality queries.

The key concept under set containment is the *restrictiveness* of the queries. That is, a query Q_1 is contained into a query Q_2 if Q_1 is more restrictive than Q_2. Under bag semantics, the key concept is the *multiplicity* with which both queries obtain any fact. It was clear that, in Example 1.1, Q_1 would not obtain more facts than Q_2; however, it obtains some facts with higher multiplicity than Q_2.

7.2 Bag containment of equality queries

Conjunctive queries under bag semantics were early studied by Dayal, Goodman and Katz in [DGK82], where they sketched an extended relational algebra with control over duplicate elimination. Later, Klausner [Kla86] studied the problem of containment and equivalence of conjunctive queries under bag semantics.

Ioannidis and Ramakrishnan [IR92, IR94] used databases whose tuples had an associated *label*, which could have different meanings. For example, the label could be a real number between 0 and 1, indicating the probability of a fact belonging to a database; this labelling systems leads to fuzzy sets. The label could also be a positive integer, where it represents the multiplicity of the fact in the database. Ioannidis and Ramakrishnan provide some results about containment for queries over databases that use bag semantics. The most important of them is a necessary and sufficient condition for the bag containment of queries that have no repeated predicate names in their bodies.

However, the first major contribution to the problem of bag containment of equality queries was introduced by Chaudhuri and Vardi [CV93], where they gave two necessary and one sufficient condition to check the containment.

The following three theorems show the two necessary and one sufficient conditions given by Chaudhuri and Vardi [CV93].

Theorem 7.1 *Let Q_1 and Q_2 be two equality queries such that $Q_1 \leq_b Q_2$. Then, for each predicate name p, the number of predicates with name p in Q_1 is less than or equal to the number of predicates with name p in Q_2.* □

Theorem 7.2 *Let Q_1 and Q_2 be two equality queries such that $Q_1 \leq_b Q_2$. Then, every predicate of Q_1 is assigned to some predicate of Q_2 by some of the possible containment mappings from Q_2 to Q_1.* □

Theorem 7.3 *Let Q_1 and Q_2 be two equality queries. If there exists an onto mapping from Q_2 to Q_1, then Q_1 is bag contained into Q_2 ($Q_1 \leq_b Q_2$). An onto mapping τ from an equality query Q_2 to another equality query Q_1 is a containment mapping from Q_2 to Q_1 such that every predicate in Q_1 is assigned to some predicate in Q_2.* □

Brisaboa and Hernández [BH97] proposed the use of a new procedure that solves the problem of testing the bag containment of equality queries, reducing it to the comparison of pairs of polynomials. This procedure is a particularization of *QCC* to test this type of containment, and will be described in Chapter 8.

7.3 Bag containment of inequality queries

To the best of our knowledge, there is no work done in this kind of containment. We shall describe in Chapter 9 how *QCC* can be applied to test the bag containment of inequality queries, offering a proof of its correctness.

7.4 Summary

This chapter has briefly reviewed the previous work about the bag containment problem of conjunctive queries. The work done by Brisaboa and Hernández [BH97] will be adapted to fit the three steps of *QCC* in Chapter 8. To the best of our knowledge, there have been no achievements in the problem of bag containment of equality queries; our own contributions for this problem will be shown in Chapter 9.

Chapter 8

Applying QCC to test bag containment of equality queries

8.1 Introduction

Under bag semantics, every fact in a database has an associated multiplicity, which is the number of copies of the fact in the database (See Section 2.3.2). For example, a database $D = \{p(A, B; [3])\}$ represents a database with 3 copies of the fact $p(A, B)$.

The bag containment of conjunctive queries is defined, as shown in Chapter 2, as

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

That is, Q_1 is bag contained in Q_2 if and only if the result of applying Q_1 to any database D is a subbag of the result of applying Q_2 to the same database D. Using QCC to test bag containment of equality queries, we only need to test the containment over the set of canonical databases built from the body of Q_1. Therefore, for a suitable test 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 [Bri97]. We shall show here how this procedure perfectly fits into the 3 steps of QCC.

The rest of this chapter first introduces the concept of Label System [IR92], and then the three steps of QCC are explained in detail. The last section proves the correctness of QCC to test bag containment of equality queries.

8.2 Label systems

In order to test the bag containment of conjunctive queries, we need to define databases (seen as bags of facts) with respect to two label systems, depending on the labels of the facts that represent their multiplicities. The first type corresponds to those databases whose facts have multiplicities represented by nonnegative integer numbers ($\mathbb{Z}^+$). The second type uses polynomials to represent the (symbolic) multiplicity of every fact. This second type of databases will be used to test the bag containment of conjunctive queries (equality as well as inequality queries).

The definition of label system given in this chapter are adapted from [IR92].

8.2.1 Preliminary definitions

A *fact* is a Horn clause with exactly one positive literal. Given a predicate p, a fact defined on p is a fact whose predicate name is p. For a fact $p(A_1, \ldots, A_n)$, p is its predicate name and $A_1, \ldots, A_n$ are its *arguments*. If all the arguments are constants, the fact is called a *ground fact*.

A *database scheme* is a finite set of predicate names. In this work, we assume that all predicates are implicitly in $\mathcal{U}$, a fixed database scheme, and that there is a fixed set of constants, the *Herbrand Universe*. The *Herbrand Base* for $\mathcal{U}$, denoted $B_{\mathcal{U}}$, is the set of all ground facts that can be formed using predicate names in $\mathcal{U}$ and the constants in $B_{\mathcal{U}}$ [Llo87].

8.2.2 Definition of label systems

A label system $\mathcal{L}$ is a quintuple $\mathcal{L} = \langle L, *, +, 0, \leq \rangle$ such that:

L is a domain of labels with a partial order $\leq$.

$*$ is a binary operation (called *product*) on L that is associative and commutative. In this work, we shall mainly use the implicit product notation ommiting the $*$ symbol, for example writing $m_1 m_2$ instead of $m_1 * m_2$.

$+$ is the *addition*, a binary operation on L that is associative and commutative.

0 is an element of L, which is the additive identity and the annihilator with respect to the product. 0 is also the least element with respect to the partial order $\leq$ defined on L. That is, $\forall a \in L,\ a + 0 = a; a * 0 = 0; and\ 0 \leq a$.

$\forall a, b \in L - \{0\},\ a \leq a * b$.

$\forall a, b, c, d \in L,\ (a \leq b \wedge c \leq d) \Rightarrow (a + c \leq b + d)$.

$\forall a \in L,\ \exists b \in L$ such that $a \leq b \wedge a \neq b$. The relationship between two elements a and b such that $a \leq b \wedge a \neq b$ is represented by the usual notation $a < b$.

For the scope of this Thesis, we shall define two label systems, denoted **LS1** and **LS2**:

LS1: A label system $\mathcal{L}$ of type LS1 is a quintuple $\mathcal{L} = \langle \mathbb{Z}^+, *, +, 0, \leq \rangle$. That is, the label domain is $\mathbb{Z}^+$, the set of nonnegative integer numbers; the product and addition operations, as well as the $\leq$ partial order, are the usual for integer numbers.

LS2: A label system $\mathcal{L}$ of type LS2 uses the polynomial arithmetic, and it is defined as the quintuple $\mathcal{L} = \langle \mathbf{P}, *, +, 0, \leq \rangle$, where $\mathbf{P}$ is a domain of polynomials whose coefficients and variables take their values from $\mathbb{Z}^+$.

The zero (0) element is the polynomial $\mathbf{0}$, that is, the polynomial that is evaluated to 0 for all values of its variables.

The $\leq$ relationship between polynomials is defined as follows. Let $P_1(\vec{X})$ and $P_2(\vec{X})$ be two polynomials. Then, $P_1(\vec{X}) \leq P_2(\vec{X})$ if and only if $P_1(\vec{X}) \leq P_2(\vec{X})$ in $\mathbb{Z}^+$. That is, for any evaluation ρ of the polynomials, $\rho(P_1(\vec{X})) \leq \rho(P_2(\vec{X}))$. An evaluation of a polynomial $P(\vec{X})$ is a function from the variables in $\vec{X}$ to $\mathbb{Z}^+$. The result of an evaluation is always a nonnegative integer.

8.2.3 Definition of databases with respect to a label system

Let $\mathcal{L} = \langle L, *. +, 0, \leq \rangle$ be a label system. A database D with respect to $\mathcal{L}$ is defined as a function from the Herbrand base $B_{\mathcal{U}}$ to L.

A database D with respect to a label system $\mathcal{L}$ is represented as a set of facts of the form

$$p(A_1, \ldots, A_n; [m]),$$

where $p(A_1, \ldots, A_n)$ is a fact in $B_{\mathcal{U}}$, and $m \in L - \{0\}$. That means that those elements of the Herbrand base that are mapped to 0 are not shown in the database.

A fact $p(a_1, \ldots, a_n)$ is in D, represented $p(a_1, \ldots, a_n) \in D$, if there is a fact $p(b_1, \ldots, b_n; [m])$ such that $a_i = b_i$ for $i = 1, \ldots, n$.

The multiplicity of the fact $t = p(b_1, \ldots, b_n)$ in D is m, denoted $|t|_D = m$, if there is a fact $p(b_1, \ldots, b_n; [m])$ in D. The multiplicity of a fact t not present in D is zero, that is, $|t|_D = 0$.

8.2.4 LS1 and LS2 databases

A database defined with respect to an LS1 label system will be denoted *LS1 database*, and it will use the integer arithmetic. That is, the $*$ and $+$ operators are the integer product and addition. The relationship $\leq$ is the usual for integers.

An *LS2 database* is a database defined with respect to an LS2 label system. The binary operators $*$ and $+$, as well as the $\leq$ relationship, are those defined for polynomials. The multiplicities in an LS2 database are said to be *symbolic* multiplicities.

Example 8.1 The following database D is an LS1 database. D' is an LS2 database.

$$D = \{r(a, b; [2]),\ r(t, j; [4])\}$$

$$D' = \{p(a, b; [m_1]),\ p(t, t; [m_2 m_3 + m_2]),\ s(a; [2m_3 + m_4 m_6^2])\}$$

□

Two databases are isomorphic is they are identical after a consistent renaming of their constants. If one (or both) databases are LS2 databases, it is necessary to assign values to the symbolic multiplicities in order to have two identical databases.

Example 8.2 Let $D = \{p(a, b; [m_1],\ \ p(b, c; [m_2]),\ \ r(a; [m_3])\}$ be an LS2 database. An LS1 database isomorphic to D is $D' = \{p(6, 7; [1],\ p(7, 9; [6]),\ r(6; [8])\}$. □

The application of a conjunctive query over a database, either LS1 or LS2, is defined in terms of assignment mappings [Ull82], as defined in Chapter 2.

8.3 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. Thus, canonical databases used to test bag containment are *LS2* databases; the multiplicities of their facts are symbolic and they use the polynomial arithmetics.

Therefore, the step (4) of Algorithm 2 is the following:

$//CDBS = \{\theta_1(db(Q_1)), \ldots, \theta_{j-1}(db(Q_1))\}$

4. Adaptation of canonical databases
   For $i = 1$ to $j - 1$
      //Add a symbolic multiplicity to
      //each fact of every canonical database
      $d_i = \{p_k(Y_{k1}, \ldots, Y_{kn}; [m_{ik}]) \mid p_k(Y_{k1}, \ldots, Y_{kn}) \in \theta_i(db(Q_1))\}$
   Return $CDBS(Q_1) = \{d_1, \ldots, d_{j-1}\}$ □

Example 8.3 Let Q_1 and Q_2 be the following equality queries.

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

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

The set of canonical databases for Q_1 are shown in Table 8.1.

□

8.4 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. If this is the case, all assignment mappings must be taken into account to compute the final multiplicity of the canonical fact, as shown in Section 2.3.2.

Table 8.1: 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

The canonical databases have symbolic multiplicities, that is, they are *LS2 databases*. Therefore, the binary operations + and $*$ (addition and product) used to compute the multiplicities are the addition and product of polynomials.

Example 8.4 (Continued from Example 8.3).

The column CDB in Tables 8.2 and 8.3 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 8.2) and Q_2 (in Table 8.3) 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 2.3.2. 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

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

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

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.

Table 8.2: Multiplicities of the canonical facts obtained by Q_1.

	CDB		t_d	Applying Q_1	
	p	r	q	Assignm. 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 8.3: Multiplicities of the canonical facts obtained by Q_2.

	CDB		t_d	Applying Q_2	
	p	r	q	Assignm. 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	$m_p m_{r1}^2+$ $m_p m_{r2}^2$

□

8.5 Step 3: Test the bag containment

We have at this point created $CDBS(Q_1)$ and computed the multiplicity of every canonical fact when 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, as we shall prove in Theorem 8.1, 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 method presented in [BH97] can be used.

Example 8.5 (Continued from Example 8.4) Notice that, for d_1 and d_4, Q_1 and Q_2 obtain the canonical fact with the same multiplicity. For d_2, d_3 and d_5, Q_2 obtains it with a higher multiplicity. Then, for all canonical databases d, $|t_d|_{Q_1(d)} \leq |t_d|_{Q_2(d)}$. Thus, for this example, $Q_1 \leq_b Q_2$. □

8.6 Validation of QCC for the bag containment of equality queries

The use of canonical databases to test the bag containment of equality queries is based on the fact that each canonical database $d \in CDBS(Q_1)$ represents (is isomorphic to) the application of an assignment mapping from Q_1 to any ground database D, as shown in the following lemma.

Lemma 8.1 *Let Q_1 be the equality query $q(\vec{W})$:- $p_1(\vec{Y_1}), \ldots, p_l(\vec{Y_l})$. Let τ be an assignment mapping from Q_1 to a database D. Let $sd = \{\tau(p_1(\vec{Y_1}; [c_1])), \cdots, \tau(p_l(\vec{Y_l}; [c_l]))\}$; i.e., sd is the subbag of D where τ maps the ordinary predicates of Q_1. Then, there exists a canonical database $d_i \in CDBS(Q_1)$ that is isomorphic to sd:*

$$\exists d_i \in CDBS(Q_1) \mid d_i \textit{ is isomorphic to } sd$$

Proof: The proof for this Lemma is practically the same as the proof of Lemma 3.1, but we have to take into account the multiplicity of the facts in sd and the symbolic multiplicities of the facts in the canonical databases.

The bag sd is the subbag of D obtained by applying τ to the body of Q_1. Note that every canonical database d is obtained by using a mapping from $db(Q_1)$, which is isomorphic to the body of Q_1, to a set of uninterpreted constants A_Q.

Assume that τ maps every variable of Q_1 to the same constant a in sd. By construction of $CDBS(Q_1)$, there exists a canonical database, say $d_1 = \theta_1(db(Q_1))$, where the Q-mapping θ_1 maps every variable of Q to the same uninterpreted constant, $A \in A_Q$. It is obvious that sd and d_1 are isomorphic, because if, in every fact $p_i(A, \ldots, A; [m_i])$ of d_1, we replace A by a and m_i by the multiplicity of the fact $p_i(a, \ldots, a)$ in sd, that is, $|p_i(a, \ldots, a)|_{sd}$, sd and d_1 become identical.

Now, assume that τ maps all variables of Q_1 to the constant a, except one, which is mapped to a different constant b. As in the previous case, there exists a canonical database built using a Q-mapping with the same pattern of equalities among two uninterpreted constants and whose facts have symbolic multiplicities.Therefore, there will be a canonical database, say d_2, which is isomorphic to sd for this case, because if, in every fact $p_i(A, \ldots, B, \ldots, A; [m_i])$ of d_2, we replace A by a, B by b and m_i by the multiplicity of the fact $p_i(a, \ldots, b, \ldots, a)$ in sd, that is, $|p_i(a, \ldots, b, \ldots, a)|_{sd}$, sd and d_2 become identical.

The same method of reasoning can be used to cover all possible patterns of equalities among the constants in sd to which the variables of Q_1 are mapped. By construction of $CDBS(Q_1)$, the equalities among the uninterpreted constants in the facts of the canonical databases cover all the possible patterns of equalities among the variables of Q_1 when they are mapped to any ground database D. Therefore, there always exists a canonical database d_i isomorphic to sd. □

The following theorem and corollary prove the validity of QCC to test query containment.

Theorem 8.1 *Given two equality queries Q_1 and Q_2, Q_1 is bag contained into Q_2 ($Q_1 \leq_b Q_2$) if and only if $\forall d \in CDBS(Q_1)$, $|t_d|_{Q_1(d)} \leq |t_d|_{Q_2(d)}$.*
Proof:

Only If: If there is a $d \in CDBS(Q_1)$ such that $|t_d|_{Q_1(d)} \not\leq |t_d|_{Q_2(d)}$, we can build a ground database D isomorphic to d that is a counterexample to the bag containment, showing that $Q_1 \not\leq_b Q_2$.

If: Assuming $|t_d|_{Q_1(d)} \leq |t_d|_{Q_2(d)}$, $\forall d \in CDBS(Q_1)$, we want to prove that $Q_1 \leq_b Q_2$.

Let D be an arbitrary database from which Q_1 obtains a fact u using several assignment mappings. By Lemma 8.1, the bag of facts reached from the atoms in Q_1 by every assignment mapping that obtains u is isomorphic to a canonical database d. Since Q_2 obtains the canonical fact t_d with at least the same multiplicity as Q_1 from all canonical databases, the total multiplicity of u obtained by Q_2 is at least the same as the multiplicity obtained by Q_1 for this fact u. Therefore, $Q_1 \leq_b Q_2$. □

Corollary 2 *Given two equality queries Q_1 and Q_2, Q_1 is bag contained into Q_2 ($Q_1 \leq_b Q_2$) if and only if $\forall d \in CDBS(Q_1)$, $Q_1(d) \subseteq_b Q_2(d)$.*
Proof:

Only If: If there is a $d \in CDBS(Q_1)$ such that $Q_1(d) \nsubseteq_b Q_2(d)$, we can build a ground database D to show a counterexample that shows that $Q_1 \nleq_b Q_2$.

If: By construction, $t_d \in Q_1(d)$; by hypothesis, $t_d \in Q_2(d)$ with at least the same multiplicity as the obtained by Q_1. Then, $\forall d \in CDBS(Q_1)$, $|t_d|_{Q_1(d)} \leq |t_d|_{Q_2(d)}$. Using the previous theorem, we conclude that $Q_1 \leq_b Q_2$.

8.7 Summary

This chapter has shown how the procedure described in [Bri97, BH97] fits the three steps of QCC. This procedure, which was the start point for this Thesis, adapts the set of canonical databases including multiplicities in their facts so they are suitable to test the containment under bag semantics. The procedure reduces the problem of testing bag containment of equality queries to the problem of comparing pairs of polynomials over $\mathbb{Z}^+$.

Chapter 9

Applying QCC to test bag containment of inequality queries

9.1 Introduction

The canonical database set for a query Q_1 used to test bag containment of inequality queries must include multiplicities in the facts, and constraints that affect the uninterpreted constants in the database.

This particular type of containment requires a more complicated treatment of the canonical databases, because the application of the assignment mappings (from either Q_1 or Q_2) to a canonical database that derive the canonical fact is not straightforward.

Under set semantics, we are interested in whether a query obtains the canonical fact or it does not obtain it, because there are no multiplicities in the facts. Therefore, we only need to know if there is (at least) *one* assignment mapping from a query to a canonical database that obtains the canonical fact. Under bag semantics, we need to apply *all* the assignment mappings from a query to a canonical database in order to compute the total multiplicity of the canonical fact derived by the query. All assignment mappings are needed because each one of them adds a monomial to the polynomial that represents the total multiplicity of the canonical fact, as seen in Chapter 8.

The process of finding all assignment mappings from an inequality query to a database is more complicated than for equality queries: All the assignment mappings from an equality query to a database can always be applied;

however, for inequality queries, the application of an assignment mapping τ from an inequality query Q to a database D must also satisfy the built-in predicates of Q, and it is not so clear when an assignment mapping can be applied, as we shall see in Example 9.1.

Taking into account these two factors (bag semantics and presence of built-in predicates), the first step of QCC for this case must adapt the canonical databases including multiplicities in their facts and constraints in the databases 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, and the third step tests the bag containment by comparing the polynomials that represent those multiplicities.

The following sections describe the three steps of QCC, as well as the proof of its correctness, to test bag containment of inequality queries.

9.2 Step 1: Build $CDBS(Q_1)$

In order to test bag containment, as in the previous chapter, canonical databases will include multiplicities in their facts. However, the most interesting aspect of the use of QCC to test bag containment of inequality queries is the management of the inequalities in the queries in conjunction with the multiplicities in the facts. In Chapter 6 we added some constraints ($constraints(d)$) to each canonical database, in order to ensure that Q_1 always obtains the canonical fact from them. However, for this type of containment, $constraints(d)$ are not restrictive enough to let us know which assignment mappings from Q_1 to each canonical database can be applied, as Example 9.1 shows.

Example 9.1 Consider the following query Q_1 and the canonical database d_1:

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

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

Note that the facts of the canonical database already have multiplicities. Also note that $constraints(d_1)$ is generated using the method presented in Chapter 6. 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:

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

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

$$m_{r1}m_{p1}m_{p2} + m_{r1}m_{p2}^2$$

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 (as it was in Chapter 8 to test the bag containment of equality queries), it is clear that $constraints(d)$, defined identically as for the test of the 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). Let us continue with the example to show how this can be done.

Example 9.2 Let us "split" the canonical database d_1 from Example 9.1 into two canonical databases that have the same bag of facts, but different constraints that specify which mappings can be applied. That is, we split d_1 into two databases d_1^1 and d_1^2 that have the same bags of facts as d_1, but with the following constraints:

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

$$constraints(d_1^2) = (A \neq B \wedge B \neq C \wedge A \neq C) \wedge A < C \wedge A < B$$

It is clear that, with these new canonical databases, the multiplicity of the canonical fact obtained by Q_1 is unique, because τ_1 and τ_4, but not τ_2 and τ_3, can be applied to d_1^1 (enforced by adding $\neg(A < B)$ to $constraints(d_1^1)$), and all 4 mappings can be applied to d_1^2 (enforced by adding $A < B$ to $constraints(d_1^2)$). Therefore,

$$|t_{d_1^1}|_{Q_1(d_1^1)} = m_{r1}m_{p1}m_{p2} + m_{r1}m_{p2}^2$$

and

$$|t_{d_1^2}|_{Q_1(d_1^2)} = m_{r1}m_{p1}m_{p2} + m_{r1}m_{p1}^2 + m_{r1}m_{p2}m_{p1} + m_{r1}m_{p2}^2.$$

□

The following definition is needed to use the assignment mappings from Q_1 to each d_i to build the final set $CDBS(Q_1)$.

Definition 9.1 $\mathcal{M}$: Possible assignment mappings from Q_1 to a canonical database d_i

Let τ_1, ..., τ_l be the set of assignment mappings that can be applied from Q_1 to a canonical database d_i, and let τ_1 be the assignment mapping isomorphic to the Q-mapping θ_i used to build d_i. Note that there always exists such an assignment mapping, and it can always be applied to d_i to derive the canonical fact. It is possible to enforce that, as it was done in the test of set containment of inequality queries in Chapter 6, by adding $\tau_1(K)$ (for each built-in predicate K in Q_1) to $constraints(d_i)$.

The remaining assignment mappings, τ_2,...,τ_l, are defined as *possible mappings*, because they may or may not be applied. It depends on whether the built-in predicates of Q_1 are satisfied with these assignment mappings. Let us denote $\mathcal{M}$ the set $\{\tau_2, \ldots, \tau_l\}$, the possible assignment mappings from Q_1 to d:

$$\mathcal{M} = \{\tau_2, \ldots, \tau_l\}$$

Depending on the "less than" and equality relationships among the uninterpreted constants in d_i, it is possible that all assignment mappings in a given subset of $\mathcal{M}$ can be applied to d_i. All such subsets are elements of $P(\mathcal{M})$ (parts of $\mathcal{M}$)

$$P(\mathcal{M}) = \{\emptyset,\ \{\tau_2\}, \ldots,\ \{\tau_l\},\ \{\tau_2, \tau_3\}, \ldots, \mathcal{M}\}$$

This set contains 2^{l-1} elements. Let us represent it as

$$P(\mathcal{M}) = \{M_1, \ldots, M_{2^{l-1}}\}$$

□

As shown in the previous example, in order to use QCC to test bag containment of inequality queries, it is necessary to split each canonical database d_i in the original $CDBS(Q_1)$ into different databases that have the same bags of facts but with different sets of constraints. Each of these final databases will allow Q_1 to derive the canonical fact using only a predefined set of assignment mappings in $P(\mathcal{M})$.

Using $P(\mathcal{M})$, we can describe completely the constraints associated to canonical databases to test bag containment of inequality queries. The formula *constraints*(d_i) is composed of three sets of constraints (the first two defined identically as in Chapter 6 to test the set containment of inequality queries).

- 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 $M_j \in P(\mathcal{M})$ can be applied, 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.

We shall refer to these constraints again as *constraints*(d) even when they are different from those defined for the use of QCC to test set containment of inequality queries. The reason is that *constraints*(d) represents, in both cases, a set of constraints that comes from Q_1 and ensures that Q_1 derives the canonical fact: in the case of bag containment, with a predefined set of assignment mappings; in the case of set containment, with at least the assignment mapping isomorphic to the Q-mapping used to build d.

It is easy to see that a canonical database d_i can be split into a maximum of 2^{l-1} canonical databases, l being the number of *possible mappings* from Q_1 to d_i, because the two first sets of constraints in *constraints*(d_i) are unique, and there are 2^{l-1} possibilities for the third set of constraints, because the cardinality of $P(\mathcal{M})$ is 2^{l-1}. Of course, all canonical databases that have an unsatisfiable *constraints*(d_i) will not be considered, because it is not possible to build a ground database isomorphic to it, as will be shown in Example 9.3.

The formal specification of the step 4 of Algorithm 2 is the following:

4. Generation of canonical databases
   For $i = 1$ to $j - 1$
   //Initially, add the multiplicities to the facts
   $d_i = \{p_c(Y_{c1}, \ldots, Y_{cn}; [m_{ck}]) \mid p_c(Y_{c1}, \ldots, Y_{cn}) \in \theta_i(db(Q_1))\}$
   $constraints(d_i) = (\theta_i(K_1) \wedge \cdots \wedge \theta_i(K_n) \ \wedge \ (A_j \neq A_k, \forall j, k \ 1 \leq j \neq k \leq q))$

   //Build the different canonical databases by adding the necessary
   //constraints

   //Let $\tau_1, \ldots, \tau_l$ be assignment mappings from Q_1 to d_k,
   //where τ_1 is isomorphic to θ_i
   $//\mathcal{M} = \{\tau_2, \ldots, \tau_l\}$
   $//P(\mathcal{M}) = \{\emptyset, \ \{\tau_2\}, \ldots, \ \{\tau_l\}, \ \{\tau_2, \tau_3\}, \ldots, \mathcal{M}\}$
   $P(\mathcal{M}) = \{M_1, \ldots, M_{2^{l-1}}\}$
   for $s = 1$ to 2^{l-1} {
   $constraints(d_i^s) = constraints(d_i)$
   $F = true$
   for $t = 2$ to l {
   if $\tau_t \in M_s$
   then $F = F \wedge \tau_t(K_1) \wedge \cdots \wedge \tau_t(K_n)$
   else $F = F \wedge \neg[\tau_t(K_1) \wedge \cdots \wedge \tau_t(K_n)]$
   }
   $constraints(d_i^s) = constraints(d_i^s) \ \wedge \ F$
   if $constraints(d_i^s)$ is unsatisfiable
   or $(\exists x, \ 1 \leq x < 2^{l-1} : d_i^x$ is isomorphic to d_i and $constraints(d_i^x) = constraints(d_i^s))$
   then $d_i^s = \emptyset$
   else // The canonical database d_i^s is generated
   $d_i^s = \theta_i(db(Q_1))$
   Associate $constraints(d_i^s)$ to d_i^s
   }

□

This step performs the particularization of Algorithm 2 needed to test bag containment of inequality queries, adding multiplicities to the facts and the necessary constraints.

Note that now each canonical database d_i^s has a subindex i and a superindex s. The subindex indicates the original canonical database d_i where d_i^s comes from, and the superindex indicates the case (refering to the set of assignment mappings always applicable to the canonical database) used to build d_i^s.

Example 9.3 (Continued from Example 9.1) The mapping τ_1 is isomorphic to θ_1, therefore it can always be applied. The rest of the mappings are the possible mappings $\mathcal{M} = \{\tau_2, \tau_3, \tau_4\}$, therefore $P(\mathcal{M}) = \{\emptyset, \{\tau_2\}, \{\tau_3\}, \{\tau_4\}, \{\tau_2, \tau_3\}, \{\tau_2, \tau_4\}, \{\tau_3, \tau_4\}, \{\tau_2, \tau_3, \tau_4\}\}$. Table 9.1 shows the 8 possibilities for the third set of constraints that will be associated to the original canonical database d_1 that produce the 8 final canonical databases d_1^0 to d_1^7.

Each case C_i in the table represents a set $M_i \in \mathcal{M}$, showing the assignment mappings that belong to M_i as a positive literal (τ_j) and those that do not belong to M_i as a negative literal ($\neg\tau_j$). For instance, the case C_2 represents the set $M_2 = \{\tau_3\}$, also identified as $\neg\tau_2\ \tau_3\ \neg\tau_4$ (meaning that $\tau_2 \notin M_2$, $\tau_3 \in M_2$, and $\tau_4 \notin M_2$).

Table 9.1: List of sets of possible mappings that can be applied

Case	Mappings	$constraints(d_1)$	Constraints to be added
C_0	$\neg\tau_2\ \neg\tau_3\ \neg\tau_4$	$(A \neq B \wedge B \neq C \wedge A \neq C) \wedge A < C$	$\neg(A < B) \wedge \neg(A < B) \wedge \neg(A < C)$
C_1	$\tau_2\ \neg\tau_3\ \neg\tau_4$	$(A \neq B \wedge B \neq C \wedge A \neq C) \wedge A < C$	$(A < B) \wedge \neg(A < B) \wedge \neg(A < C)$
C_2	$\neg\tau_2\ \tau_3\ \neg\tau_4$	$(A \neq B \wedge B \neq C \wedge A \neq C) \wedge A < C$	$\neg(A < B) \wedge (A < B) \wedge \neg(A < C)$
C_3	$\neg\tau_2\ \neg\tau_3\ \tau_4$	$(A \neq B \wedge B \neq C \wedge A \neq C) \wedge A < C$	$\neg(A < B) \wedge \neg(A < B) \wedge (A < C)$
C_4	$\tau_2\ \tau_3\ \neg\tau_4$	$(A \neq B \wedge B \neq C \wedge A \neq C) \wedge A < C$	$(A < B) \wedge (A < B) \wedge \neg(A < C)$
C_5	$\tau_2\ \neg\tau_3\ \tau_4$	$(A \neq B \wedge B \neq C \wedge A \neq C) \wedge A < C$	$(A < B) \wedge \neg(A < B) \wedge (A < C)$
C_6	$\neg\tau_2\ \tau_3\ \tau_4$	$(A \neq B \wedge B \neq C \wedge A \neq C) \wedge A < C$	$\neg(A < B) \wedge (A < B) \wedge (A < C)$
C_7	$\tau_2\ \tau_3\ \tau_4$	$(A \neq B \wedge B \neq C \wedge A \neq C) \wedge A < C$	$(A < B) \wedge (A < B) \wedge (A < C)$

Note that for the cases C_0, C_1, C_2, C_4, C_5, and C_6, the resulting formula $constraints(d_1^s)$ is unsatisfiable. Therefore, these cases can be discarded. Only 2 canonical databases (that correspond to the cases C_3 and C_7, therefore labelled d_1^3 and d_1^7), shown in Table 9.2, produce satisfiable formulas and can be considered to test the bag containment.

Table 9.2: Splitting of a canonical database into 2

Name	Q-Mapping	$\theta_1(db(Q_1))$		$constraints(d_1)$	t_d
		r	p		q
d_1^3	θ_1	AA $[m_{r1}]$	AB $[m_{p1}]$ AC $[m_{p2}]$	$[A \neq B \wedge B \neq C \wedge A \neq C \wedge A < C] \ \wedge\ [\neg(A < B)]$	AA
d_1^7	θ_1	AA $[m_{r1}]$	AB $[m_{p1}]$ AC $[m_{p2}]$	$[A \neq B \wedge B \neq C \wedge A \neq C \wedge A < C] \ \wedge\ [(A < B)]$	AA

□

9.3 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. Both queries will derive it with a multiplicity that is represented by a polynomial (recall that canonical databases are LS2 databases).

9.3.1 Application of Q_1 to a canonical database d_i^s

By construction of $CDBS(Q_1)$, the assignment mappings from Q_1 that can be applied to a canonical database d_i^s are known. Therefore, the multiplicity of the canonical fact $t_{d_i^s}$ obtained by Q_1 is computed by adding the multiplicities obtained by each single mapping, as shown in Chapter 2.

9.3.2 Application of Q_2 to a canonical database d_i^s

Similiarly as for Q_1, there will be different sets of assignment mappings that can be applied from Q_2 to d_i^s such that Q_2 obtains the canonical fact. For each of these sets of mappings, 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_i^s (to obtain the canonical fact) can be applied.

The procedure used to compute all the different multiplicities is the following:

Let $\mathcal{M}' = \{\tau_1, \ldots, \tau_m\}$ be the set of assignment mappings from Q_2 to d_i^s (*possible mappings*). The sets of possible assignment mappings that can be applied to d_i^s 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 build the list of cases that correspond to the different sets of mappings that can be applied from Q_2 to d_i^s. For each case, we shall build a formula that is the conjunction of $constraints(d_i^s)$ and other constraints that specify that a concrete set of assignment mappings M'_j in $P(\mathcal{M}')$ can be applied from Q_2 to d_i^s. 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 in order for the case to be considered. If the formula is unsatisfiable, it would mean that Q_2 cannnot be applied to d_i^s to 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 $(|t_{d_i^s}|_{Q_2(d_i^s)})_{M'_j}$.

Note that each of the previous cases is a *possible* case, that is, for some values of the uninterpreted constants, Q_2 can obtain the canonical fact using the sets of assignment mappings specified for each case. However, there is no guarantee that a particular set of mappings can always be applied. We need to compute the multiplicity of the canonical fact for all the possible cases because, as we shall see in the next section, the multiplicity obtained by Q_1 must be compared with all of them in order to prove the bag containment.

A specially important case is C_0, which 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_i^s can be applied, so Q_2 does not derive the canonical fact (the multiplicity would be 0)[1]. If this is the case, the general QCC procedure can stop at this point concluding that the bag containment does not hold.

The following example shows how to apply Q_2 to a canonical database.

Example 9.4 Assume there is the following canonical database d_1^0 for some query Q_1 defined over two predicates r and p:

r	p	$t_{d_1^0}$	$constraints(d_1^0)$
A B $[m_{r1}]$	A B $[m_{p1}]$ D C $[m_{p2}]$	A	$A \neq B \wedge A \neq C \wedge A \neq D \wedge B \neq C \wedge B \neq D \wedge C \neq D$ $\wedge A < C$

Let Q_2 be the query

$$Q_2: \; q(X) \text{ :- } r(X,Y), p(X,Y), p(Z,T), X < T$$

There are two assignment mappings from Q_2 to d_1^0 that obtain the canonical fact. They are shown in the following table, along with the constraints they must satisfy and the multiplicity of the canonical fact obtained using only each individual mapping:

[1] Note that the formula for the case C_0 is identical to the formula F used in Chapter 6 to test set containment of inequality queries, where the satisfiability of F also meant that Q_2 would not obtain the canonical fact.

Name	X Y Z T	r(X,Y)	p(X,Y)	p(Z,T)	$X < T$	$(\lvert t_{d_1^0} \lvert_{Q_2(d_1^0)})_{\tau_j}$
τ_1	A B A B	$r(A,B;[m_{r1}])$	$p(A,B;[m_{p1}])$	$p(A,B;[m_{p1}])$	$A < B$	$m_{r1}m_{p1}^2$
τ_2	A B D C	$r(A,B;[m_{r1}])$	$p(A,B;[m_{p1}])$	$p(D,C;[m_{p2}])$	$A < C$	$m_{p1}m_{p1}m_{p2}$

The set of possible mappings is $\mathcal{M}' = \{\tau_1, \tau_2\}$, therefore

$$P(\mathcal{M}') = \{\emptyset,\ \{\tau_1\},\ \{\tau_2\},\ \{\tau_1, \tau_2\}\}.$$

The following table shows all the possible cases, the formula that must be satisfiable in order to apply the set of mappings, and the multiplicity of the canonical fact obtained by Q_2 using only this set of mappings.

For example, the row for the case C_1 corresponds to the set of mappings $M_1' = \{\tau_1\}$, where τ_1 can be applied and τ_2 cannot (represented as $\tau_1\ \neg\tau_2$).

Case	Mappings	Formula	$(\lvert t_{d_1^0} \lvert_{Q_2(d_1^0)})_{M_j'}$
C_0	$\neg\tau_1\ \neg\tau_2$	$[A \neq B \wedge A \neq C \wedge A \neq D \wedge B \neq C \wedge B \neq D \wedge C \neq D]$ $\wedge[A < C]$ $\wedge[\neg(A < B) \wedge \neg(A < C)]$	0
C_1	$\tau_1\ \neg\tau_2$	$[A \neq B \wedge A \neq C \wedge A \neq D \wedge B \neq C \wedge B \neq D \wedge C \neq D]$ $\wedge[A < C]$ $\wedge[(A < B) \wedge \neg(A < C)]$	$m_{r1}m_{p1}^2$
C_2	$\neg\tau_1\ \tau_2$	$[A \neq B \wedge A \neq C \wedge A \neq D \wedge B \neq C \wedge B \neq D \wedge C \neq D]$ $\wedge[A < C]$ $\wedge[\neg(A < B) \wedge (A < C)]$	$m_{p1}m_{p1}m_{p2}$
C_3	$\tau_1\ \tau_2$	$[A \neq B \wedge A \neq C \wedge A \neq D \wedge B \neq C \wedge B \neq D \wedge C \neq D]$ $\wedge[A < C]$ $\wedge[(A < B) \wedge (A < C)]$	$m_{r1}m_{p1}^2+$ $m_{p1}m_{p1}m_{p2}$

Case C_0 produces an unsatisfiable formula, so it will be discarded. This implies that it is impossible that none of the two mappings from Q_2 to d_1^0 can be applied. Thus, Q_2 will always obtain the canonical fact.

The formula for the case C_1 is unsatisfiable, so it will not be considered. It means that it is not possible for Q_2 to obtain the canonical fact using *only* the assignment mapping τ_1.

The formula for the case C_2 is satisfiable. For this case, the set of mappings that can be applied is $\{\tau_2\}$, and the multiplicity of the canonical fact is $m_{r1}m_{p1}m_{p2}$.

The formula for the case C_3 is satisfiable. For this case, the set of mappings that can be applied is $\{\tau_1, \tau_2\}$, and the multiplicity of the canonical fact using this set of mappings is $m_{r1}m_{p1}^2 + m_{r1}m_{p1}m_{p2}$.

Therefore, for this example, Q_2 can obtain the canonical fact with two different multiplicities, those that correspond to the cases C_2 and C_3. In other words, applying Q_2 to any ground database isomorphic to d_1^0, and depending on the "less than" relationships among the constants, Q_2 can obtain the canonical fact either using only the assignment mapping τ_2 (therefore the multiplicity of the canonical fact is $m_{r1}m_{p1}m_{p2}$) or using both τ_1 and τ_2, with a multiplicity of $m_{r1}m_{p1}^2 + m_{r1}m_{p1}m_{p2}$. □

9.4 Step 3: Test the bag containment

At this point, the multiplicity of the canonical fact obtained by Q_1, which is predefined for each canonical database, is known. For Q_2, there are several possible multiplicities, depending on the sets of assignment mappings that can be applied. We also know that Q_2 always obtains the canonical fact with some multiplicity, by testing the unsatisfiability of the formula corresponding to case C_0 (if it were satisfiable, that would mean that the containment does not hold, because there would be cases where none of the assignment mappings from Q_2 could be applied).

In this third step, the multiplicity of the canonical fact obtained by Q_1 is compared with all the multiplicities obtained by Q_2. If, for all the 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 ($Q_1 \leq_b Q_2$), otherwise $Q_1 \not\leq_b Q_2$.

The test is done, as we shall prove in Theorem 9.1, by checking the following: $Q_1 \leq_b Q_2 \Longleftrightarrow \forall d_i \in CDBS(Q_1)\ |t_{d_i}|_{Q_1(d_i)} \leq (|t_{d_i}|_{Q_2(d_i)})_{M'_j}$, $\forall M'_j$ possible set of assignment mappings from Q_2 to d_i.

Note that (as it was the case in Example 9.4) the different multiplicities of the canonical fact obtained by Q_2 can be comparable (the multiplicity obtained in the case C_2 is strictly lower than the multiplicity for the case C_3), so in this step we need to compare the multiplicity of the canonical fact obtained by Q_1 with the lowest multiplicity obtained by Q_2. If the Q_2 obtains the canonical fact with multiplicities that are not comparable, the multiplicity obtained by Q_1 (which is unique) must be compared with all the multiplicities obtained by Q_2.

9.5 Validation of QCC for the bag containment of inequality queries

The following lemma proves, as in the previous cases, that the application of an assignment mapping from an inequality query Q_1 to any ground database D is isomorphic to some canonical database $d_i^s \in CDBS(Q_1)$.

Lemma 9.1 *Let Q_1 be an inequality query of the form $q(\vec{W})$:- $p_1(\vec{Y_1}), \ldots, p_l(\vec{Y_l}), K_1, \ldots, K_n$. Let τ be an assignment map-*

ping from Q_1 to a database D. Let $sd = \{\tau(p_1(\vec{Y_1};[c_1])), \cdots, \tau(p_l(\vec{Y_l};[c_l]))\}$; i.e., sd is the subbag of D where τ maps the ordinary predicates of Q_1. Then sd is isomorphic to a canonical database $d_i^s \in CDBS(Q_1)$:

$$\exists d_i^s \in CDBS(Q_1) \mid d_i^s \textit{ is isomorphic to } sd$$

Proof:

The first part of the proof for this lemma is similar to Lemma 8.1. Given that the canonical databases represent all the patterns of equalities among uninterpreted constants (these patterns are built by construction of the canonical databases), *sd* is isomorphic to a canonical database $d_i^s \in CDBS(Q_1)$.

Besides, in order for the isomorphism to hold, the constants in the facts of *sd* must satisfy $constraints(d_i^s)$. But this is also true, because $constraints(d_i^s)$ always represent the application of an assignment mapping from Q_1 to a ground database D (recall that, by construction of $CDBS(Q_1)$, every set of mappings that can be applied is specified by adding some specific constraints). □

The following theorem demonstrates that the test shown in the previous section, that is, $\forall d_i^s \in CDBS(Q_1)$, $|t_{d_i^s}|_{Q_1(d_i^s)} \leq (|t_{d_i^s}|_{Q_2(d_i^s)})_{M'_j}$, for all possible set of assignment mappings M'_j from Q_2 to d_i^s, is a necessary and sufficient condition to test the bag containment.

Theorem 9.1 *Let Q_1 and Q_2 of the form*

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

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

Then, $Q_1 \leq_b Q_2 \iff \forall d_i^s \in CDBS(Q_1)\ |t_{d_i^s}|_{Q_1(d_i^s)} \leq (|t_{d_i^s}|_{Q_2(d_i^s)})_{M'_j}$, for all possible set of assignment mappings M'_j from Q_2 to d_i^s.

Proof:

ONLY IF : If there is a $d_i \in CDBS(Q_1)$ such that $|t_{d_i^s}|_{Q_1(d_i^s)} \not\leq (|t_{d_i^s}|_{Q_2(d_i^s)})_{M'_j}$, for some set of assignment mappings M'_j from Q_2 to d_i^s, we can build a ground database D to show a counterexample showing that Q_2 cannot obtain a fact ($t_{d_i^s}$) with at least the same multiplicity as Q_1, and therefore $Q_1 \not\leq_b Q_2$. Note that this includes the case when

Q_2 does not obtain the canonical fact (when the case C_0 produces a satisfiable formula), because the multiplicity of the canonical fact obtained by Q_2 would be zero.

IF : Assuming $|t_{d_i^s}|_{Q_1(d_i^s)} \leq (|t_{d_i^s}|_{Q_2(d_i^s)})_{M'_j}$, for all possible set of assignment mappings M'_j from Q_2 to d_i^s, we want to prove that $Q_1 \leq_b Q_2$.

Let D be an arbitrary database from which Q_1 obtains a fact u using several assignment mappings. By Lemma 9.1, the bag of facts reached from the atoms in Q_1 by every assignment mapping is isomorphic to a canonical database d_i^s. Since Q_2 always obtains the canonical fact $t_{d_i^s}$ with at least the same multiplicity as Q_1 from all canonical databases, the total multiplicity of u obtained by Q_2 is at least the same as the multiplicity obtained by Q_1. Therefore, $Q_1 \leq_b Q_2$.

□

Corollary 3 *Given two inequality queries Q_1 and Q_2, Q_1 is bag contained into Q_2 ($Q_1 \leq_b Q_2$) if and only if $\forall d \in CDBS(Q_1),\ Q_1(d) \subseteq_b Q_2(d)$.*

Proof:

ONLY IF: If there is a $d \in CDBS(Q_1)$ such that $Q_1(d) \not\subseteq_b Q_2(d)$, we can build a ground database D to show a counterexample that shows that $Q_1 \not\leq_b Q_2$.

IF: By construction, $t_d \in Q_1(d)$; by hypothesis, $t_d \in Q_2(d)$ with at least the same multiplicity as the obtained by Q_1. Then, $\forall d \in CDBS(Q_1),\ |t_d|_{Q_1(d)} \leq |t_d|_{Q_2(d)}$. Using the previous theorem, we conclude that $Q_1 \leq_b Q_2$. □

9.6 Example

Example 9.5 Let Q_1 and Q_2 be the following inequality queries:

$$Q_1 : \ q(X,T) \ \text{:-} \ p(X), r(Y,X), r(Z,T), s(T), X > Y.$$

$$Q_2 : \ q(X,T) \ \text{:-} \ p(X), r(Y,Z), r(W,Z), s(T), Y < Z, Y \leq W.$$

Let us test if Q_1 is bag contained in Q_2, following the three steps of QCC:

Build $CDBS(Q_1)$:

Table 9.3 shows the initial set of canonical databases.

This table includes, for each initial canonical database, the two first sets of constraints that will be included in *constraints*(d), since these sets will

Table 9.3: Initial $CDBS(Q_1)$

Q-Mapping		$\theta_i(db(Q_1))$			constraints	t_d
CDB	X Y Z T	p	r	s		q
d_1	A A A A	A	AA	A	$A > A$ (Not satisfiable)	AA
d_2	A A A B	A	AA AB	B	$A \neq B \wedge A > A$ (Not satisfiable)	AB
d_3	A A B A	A	AA BA	A	$A \neq B \wedge A > A$ (Not satisfiable)	AA
d_4	A B A A	A	BA AA	A	$A \neq B \wedge A > B$	AA
d_5	B A A A	B	AB AA	A	$A \neq B \wedge B > A$	BA
d_6	A A B B	A	AA BB	B	$A \neq B \wedge A > A$ (Not satisfiable)	AB
d_7	A B A B	A	BA AB	B	$A \neq B \wedge A > B$	AB
d_8	A B B A	A	BA	A	$A \neq B \wedge A > B$	AA
d_9	A A B C	A	AA BC	C	$A \neq B \wedge A \neq C \wedge B \neq C \wedge A > A$ (Not satisfiable)	AC
d_{10}	A B A C	A	BA AC	C	$A \neq B \wedge A \neq C \wedge B \neq C \wedge A > B$	AC
d_{11}	A B C A	A	BA CA	A	$A \neq B \wedge A \neq C \wedge B \neq C \wedge A > B$	AA
d_{12}	B A A C	B	AB AC	C	$A \neq B \wedge A \neq C \wedge B \neq C \wedge B > A$	BC
d_{13}	B A C A	B	AB CA	A	$A \neq B \wedge A \neq C \wedge B \neq C \wedge B > A$	BA
d_{14}	B C A A	B	CB AA	A	$A \neq B \wedge A \neq C \wedge B \neq C \wedge B > C$	BA
d_{15}	A B C D	A	BA CD	D	$A \neq B \wedge A \neq C \wedge A \neq D \wedge B \neq C \wedge B \neq D \wedge C \neq D \wedge A > B$	AD

be included in the final constraints. The constraints generated for the Q-mappings τ_1, τ_2, τ_3, τ_6, and τ_9 already produce an unsatisfiable formula, therefore Q_1 does not derive the canonical fact from them. Thus, the corresponding canonical databases will not be further considered to test the bag containment. For the rest of the canonical databases, we must find the possible mappings from Q_1 to them, build the list of cases corresponding to the sets of possible mappings, and generate the final canonical databases.

d_4: There are two assignment mappings from Q_1 to d_4, shown in the following table.

	X Y Z T	p(X)	r(Y,X)	r(Z,T)	s(T)	$X > Y$
τ_1	A B A A	p(A)	r(B,A)	r(A,A)	s(A)	$A > B$
τ_2	A B B A	p(A)	r(B,A)	r(B,A)	s(A)	$A > B$

The assignment mapping τ_1 is isomorphic to the Q-mapping θ_4, therefore it can always be applied. The set of possible mappings for this case is $\mathcal{M} = \{\tau_2\}$, therefore $P(\mathcal{M}) = \{\emptyset, \{\tau_2\}\}$.

Let us build the list of cases that correspond to each set of possible mappings. For each case, the following table shows the constraints that would be associated to each canonical database, represented as $constraints(d_i^j)$, where i references the initial canonical database d_i and j references the case C_j.

Case	Mappings	$constraints(d_4^j)$
C_0	$\neg\tau_2$	$[A \neq B \wedge A > B] \ \wedge \ \neg(A > B)$
C_1	τ_2	$[A \neq B \wedge A > B] \ \wedge \ (A > B)$

It is easy to check that $constraints(d_4^0)$ is unsatisfiable. Therefore, only one canonical database, d_4^1, is generated,

d_5: There is only one assignment mapping from Q_1 to d_5, and it is isomorphic to θ_5. Therefore, only one canonical database, d_5^0 (which is identical to d_5 is generated.

d_7: There is only one assignment from Q_1 to d_7, isomorphic to θ_7. Therefore, only the canonical database d_7^0, identical to d_7, is generated.

d_8: Again, there is only one assignment mapping from Q_1 to d_8, and only the canonical database d_8^0 is generated.

d_{10}: As in the previous cases, the only assignment mapping from Q_1 to d_{10} is isomorphic to the Q-mapping (θ_{10} in this case), and the canonical database d_{10}^0, identical to d_{10}, is generated.

d_{11}**:** There are 4 assignment mappings that can be applied from Q_1 to d_{11}, shown in the following table.

	X Y Z T	p(X)	r(Y,X)	r(Z,T)	s(T)	$X > Y$
τ_1	A B C A	p(A)	r(B,A)	r(C,A)	s(A)	$A > B$
τ_2	A B B A	p(A)	r(B,A)	r(B,A)	s(A)	$A > B$
τ_3	A C C A	p(A)	r(C,A)	r(C,A)	s(A)	$A > C$
τ_4	A C B A	p(A)	r(C,A)	r(B,A)	s(A)	$A > C$

The assignment mapping τ_1 is isomorphic to θ_{11} and can always be applied. Thus, the set of possible assignment mappings is $\mathcal{M} = \{\tau_2, \tau_3, \tau_4\}$.

The following table shows the list of cases that correspond to each of the 8 sets of assignment mappings in $P(\mathcal{M})$.

Case	Mappings	$constraints(d_{11}^j)$
C_0	$\neg\tau_2\ \neg\tau_3\ \neg\tau_4$	$[A \neq B \wedge A \neq C \wedge B \neq C \wedge A > B] \wedge$ $\neg(A > B) \wedge \neg(A > C) \wedge \neg(A > C)$
C_1	$\neg\tau_2\ \neg\tau_3\ \tau_4$	$[A \neq B \wedge A \neq C \wedge B \neq C \wedge A > B] \wedge$ $\neg(A > B) \wedge \neg(A > C) \wedge (A > C)$
C_2	$\neg\tau_2\ \tau_3\ \neg\tau_4$	$[A \neq B \wedge A \neq C \wedge B \neq C \wedge A > B] \wedge$ $\neg(A > B) \wedge (A > C) \wedge \neg(A > C)$
C_3	$\tau_2\ \neg\tau_3\ \neg\tau_4$	$[A \neq B \wedge A \neq C \wedge B \neq C \wedge A > B] \wedge$ $(A > B) \wedge \neg(A > C) \wedge \neg(A > C)$
C_4	$\neg\tau_2\ \tau_3\ \tau_4$	$[A \neq B \wedge A \neq C \wedge B \neq C \wedge A > B] \wedge$ $\neg(A > B) \wedge (A > C) \wedge (A > C)$
C_5	$\tau_2\ \neg\tau_3\ \tau_4$	$[A \neq B \wedge A \neq C \wedge B \neq C \wedge A > B] \wedge$ $(A > B) \wedge \neg(A > C) \wedge (A > C)$
C_6	$\tau_2\ \tau_3\ \neg\tau_4$	$[A \neq B \wedge A \neq C \wedge B \neq C \wedge A > B] \wedge$ $(A > B) \wedge (A > C) \wedge \neg(A > C)$
C_7	$\tau_2\ \tau_3\ \tau_4$	$[A \neq B \wedge A \neq C \wedge B \neq C \wedge A > B] \wedge$ $(A > B) \wedge (A > C) \wedge (A > C)$

It is clear that the only satisfiable formulas are those for cases C_3 and C_7. Therefore, two canonical databases, d_{11}^3 and d_{11}^7, are generated. Both databases have the same facts as d_{11}, and their constraints are those shown in the previous table for cases C_3 and C_7, respectively. For d_{11}^3, only assignment mappings τ_1 and τ_2 can be applied; for d_{11}^7, all 4 assignment mappings can.

d_{12}**,** d_{13}**,** d_{14}**, and** d_{15}**:** There is only one assignment mapping from Q_1 to each of those canonical databases, and it is always isomorphic to the

Table 9.4: Final $CDBS(Q_1)$

Q-Mapping		$\theta_i(db(Q_1))$			$constraints(d_i^j)$	t_d
CDB	X Y Z T	p	r	s		q
d_4^1	A B A A	A$[m_p]$	BA $[m_{r1}]$ AA $[m_{r2}]$	A $[m_s]$	$A \neq B \wedge A > B$	AA
d_5^0	B A A A	B$[m_p]$	AB $[m_{r1}]$ AA $[m_{r2}]$	A $[m_s]$	$A \neq B \wedge B > A$	BA
d_7^0	A B A B	A $[m_p]$	BA $[m_{r1}]$ AB $[m_{r2}]$	B $[m_s]$	$A \neq B \wedge A > B$	AB
d_8^0	A B B A	A $[m_p]$	BA $[r]$	A $[m_s]$	$A \neq B \wedge A > B$	AA
d_{10}^0	A B A C	A $[m_p]$	BA $[m_{r1}]$ AC $[m_{r2}]$	C $[m_s]$	$A \neq B \wedge A \neq C \wedge B \neq C \wedge A > B$	AC
d_{11}^3	A B C A	A $[m_p]$	BA $[m_{r1}]$ CA $[m_{r2}]$	A $[m_s]$	$A \neq B \wedge A \neq C \wedge B \neq C \wedge A > B \wedge \neg(A > C)$	AA
d_{11}^7	A B C A	A $[m_p]$	BA $[m_{r1}]$ CA $[m_{r2}]$	A $[m_s]$	$A \neq B \wedge A \neq C \wedge B \neq C \wedge A > B \wedge (A > C)$	AA
d_{12}^0	B A A C	B $[m_p]$	AB $[m_{r1}]$ AC $[m_{r2}]$	C $[m_s]$	$A \neq B \wedge A \neq C \wedge B \neq C \wedge B > A$	BC
d_{13}^0	B A C A	B $[m_p]$	AB $[m_{r1}]$ CA $[m_{r2}]$	A $[m_s]$	$A \neq B \wedge A \neq C \wedge B \neq C \wedge B > A$	BA
d_{14}^0	B C A A	B $[m_p]$	CB $[m_{r1}]$ AA $[m_{r2}]$	A $[m_s]$	$A \neq B \wedge A \neq C \wedge B \neq C \wedge B > C$	BA
d_{15}^0	A B C D	A $[m_p]$	BA $[m_{r1}]$ CD $[m_{r2}]$	D $[m_s]$	$A \neq B \wedge A \neq C \wedge A \neq D \wedge B \neq C \wedge$ $B \neq D \wedge C \neq D \wedge A > B$	AD

corresponding Q-mapping, so only the one canonical database is generated for each case (d_{12}^0, d_{13}^0, d_{14}^0, and d_{15}^0).

Therefore, the final set of canonical databases $CDBS(Q_1)$ is the one shown in Table 9.4.

Apply Q_1 and Q_2 to each canonical database:

All the mappings from Q_1 to each canonical database that obtain the canonical fact are known. Therefore, the multiplicity of the canonical fact is the sum of the multiplicities obtained by each individual assignment mappings. Table 9.5 shows the assignment mappings from Q_1 to each canonical database and the multiplicity of the canonical fact.

In order to apply Q_2 to all canonical databases, we must find all the *possible mappings* from Q_2 to each one of them, and then build the list of cases that will show the different multiplicities of the canonical fact obtained by Q_2.

d_4^1: There are 2 possible assignment mappings that can be applied from Q_2 to d_4^1, shown in the following table.

Table 9.5: Application of Q_1 to all canonical databases

	$\theta_i(db(Q_1))$			$constraints(d_i^j)$	t_d	Application of Q_1	
CDB	p	r	s		q	Ass. mappings	$\|t_d\|_{Q_1(d)}$
						X Y Z T	
d_4^1	A$[m_p]$	BA $[m_{r1}]$ AA $[m_{r2}]$	A $[m_s]$	$A \neq B \wedge A > B$	AA	A A B A A B B A	$m_p m_{r2} m_{r1} m_s +$ $m_p m_{r1}^2 m_s$
d_5^0	B$[m_p]$	AB $[m_{r1}]$ AA $[m_{r2}]$	A $[m_s]$	$A \neq B \wedge B > A$	BA	B A A A	$m_p m_{r1} m_{r2} m_s$
d_7^0	A $[m_p]$	BA $[m_{r1}]$ AB $[m_{r2}]$	B $[m_s]$	$A \neq B \wedge A > B$	AB	A B A B	$m_p m_{r1} m_{r2} m_s$
d_8^0	A $[m_p]$	BA $[m_r]$	A $[m_s]$	$A \neq B \wedge A > B$	AA	A B B A	$m_p m_{r1} m_{r2} m_s$
d_{10}^0	A $[m_p]$	BA $[m_{r1}]$ AC $[m_{r2}]$	C $[m_s]$	$A \neq B \wedge A \neq C \wedge B \neq C \wedge$ $A > B$	AC	A B A C	$m_p m_{r1} m_{r2} m_s$
d_{11}^3	A $[m_p]$	BA $[m_{r1}]$ CA $[m_{r2}]$	A $[m_s]$	$A \neq B \wedge A \neq C \wedge B \neq C \wedge$ $A > B \wedge \neg(A > C)$	AA	A B C A A B B A	$m_p m_{r1} m_{r2} m_s +$ $m_p m_{r1}^2 m_s$
d_{11}^7	A $[m_p]$	BA $[m_{r1}]$ CA $[m_{r2}]$	A $[m_s]$	$A \neq B \wedge A \neq C \wedge B \neq C \wedge$ $A > B \wedge (A > C)$	AA	A B C A A B B A A C C A A C B A	$m_p m_{r1} m_{r2} m_s +$ $m_p m_{r1}^2 m_s +$ $m_p m_{r2}^2 m_s +$ $m_p m_{r2} m_{r1} m_s$
d_{12}^0	B $[m_p]$	AB $[m_{r1}]$ AC $[m_{r2}]$	C $[m_s]$	$A \neq B \wedge A \neq C \wedge B \neq C \wedge$ $B > A$	BC	B A A C	$m_p m_{r1} m_{r2} m_s$
d_{13}^0	B $[m_p]$	AB $[m_{r1}]$ CA $[m_{r2}]$	A $[m_s]$	$A \neq B \wedge A \neq C \wedge B \neq C \wedge$ $B > A$	BA	B A C A	$m_p m_{r1} m_{r2} m_s$
d_{14}^0	B $[m_p]$	CB $[m_{r1}]$ AA $[m_{r2}]$	A $[m_s]$	$A \neq B \wedge A \neq C \wedge B \neq C \wedge$ $B > C$	BA	B C A A	$m_p m_{r1} m_{r2} m_s$
d_{15}^0	A $[m_p]$	BA $[m_{r1}]$ CD $[m_{r2}]$	D $[m_s]$	$A \neq B \wedge A \neq C \wedge A \neq D \wedge B \neq C \wedge$ $B \neq D \wedge C \neq D \wedge A > B$	AD	A B C D	$m_p m_{r1} m_{r2} m_s$

	X Y Z T W	p(X)	r(Y,Z)	r(W,Z)	s(T)	$Y < Z$	$Y \leq W$	$(\|t_{d_4^1}\|_{Q_2(d_4^1)})_{\{\tau_i\}}$
τ_1	A B A A B	p(A; $[m_p]$)	r(B,A; $[m_{r1}]$)	r(B,A; $[m_{r1}]$)	s(A; $[m_s]$)	$B < A$	$B \leq B$	$m_p m_{r1}^2 m_s$
τ_2	A A A A A	p(A; $[m_p]$)	r(B,A; $[m_{r1}]$)	r(A,A; $[m_{r2}]$)	s(A; $[m_s]$)	$B < A$	$B \leq A$	$m_p m_{r1} m_{r2} m_s$

The set of possible mappings from Q_2 do d_4^1 is $\mathcal{M}' = \{\tau_1, \tau_2\}$, thus $P(\mathcal{M}) = \{\emptyset, \{\tau_1\}, \{\tau_2\}, \{\tau_1, \tau_2\}\} = \{M'_0, M'_1, M'_2, M'_3\}$. The following table shows whether each of these sets of assignment mappings can be applied to d_4^1.

Case	Mappings	Formula
C_0	$\neg\tau_1\neg\tau_2$	$[A \neq B \wedge A > B] \ \wedge\ \neg[(B < A) \wedge (B \leq B)] \ \wedge\ \neg[(B < A) \wedge (B \leq A)]$
C_1	$\neg\tau_1\tau_2$	$[A \neq B \wedge A > B] \ \wedge\ \neg[(B < A) \wedge (B \leq B)] \ \wedge\ [(B < A) \wedge (B \leq A)]$
C_2	$\tau_1\neg\tau_2$	$[A \neq B \wedge A > B] \ \wedge\ [(B < A) \wedge (B \leq B)] \ \wedge\ \neg[(B < A) \wedge (B \leq A)]$
C_3	$\tau_1\tau_2$	$[A \neq B \wedge A > B] \ \wedge\ [(B < A) \wedge (B \leq B)] \ \wedge\ [(B < A) \wedge (B \leq A)]$

The formula for the case C_0 is not satisfiable. This case corresponds to the empty set of assignment mappings, that is, the case when neither τ_1 nor τ_2 can be applied (it would be the case when Q_2 does not derive the canonical fact). Since the formula is not satisfiable, it means that Q_2 always obtains the canonical fact. Cases C_2 and C_3 also produce unsatisfiable formulas, therefore it is not possible to apply *exclusively* either τ_1 or τ_2. C_3 produces a satisfiable formula, therefore it is always possible to apply both mappings to d_4^1. The resulting multiplicitiy would be $|t_{d_4^1}|_{Q_1(d_4^1)} = m_p m_{r1}^2 m_s$ + $m_p m_{r1} m_{r2} m_s$.

d_5^0**:** There is only one assignment mapping from Q_2 to d_5^0, shown in the following table.

	X Y Z T W	p(X)	r(Y,Z)	r(W,Z)	s(T)	$Y < Z$	$Y \leq W$	$(\|t_{d_5^0}\|_{Q_2(d_5^0)})_{\{\tau_i\}}$
τ_1	B A B A A	p(B; $[m_p]$)	r(A,B; $[m_{r1}]$)	r(A,B; $[m_{r1}]$)	s(A; $[m_s]$)	$A < B$	$A \leq A$	$m_p m_{r1}^2 m_s$

The set of possible mappings for this case is $\mathcal{M}' = \{\tau_1\}$, therefore $P(\mathcal{M}) = \{\emptyset, \{\tau_1\}\}$. The list of cases is shown in the following table.

Case	Mappings	Formula
C_0	$\neg\tau_1$	$[A \neq B \wedge B > A] \ \wedge \ \neg[A < B \wedge (A \leq A)]$
C_1	$\neg\tau_1$	$[A \neq B \wedge B > A] \ \wedge \ [A < B \wedge (A \leq A)]$

The unsatisfiability of the formula for case C_0 indicates that Q_2 always obtains the canonical fact. Case C_1 produces a satisfiable formula, therefore τ_1 can always be applied. The multiplicity of the canonical fact obtained by Q_2 for this database is $|t_{d_5^0}|_{Q_1(d_5^0)} = m_p m_{r1}^2 m_s$.

d_7^0**:** There is only one assignment mapping from Q_2 to d_7^0, shown in the following table.

	X Y Z T W	p(X)	r(Y,Z)	r(W,Z)	s(T)	$Y < Z$	$Y \leq W$	$(\|t_{d_7^0}\|_{Q_2(d_7^0)})_{\{\tau_i\}}$
τ_1	A B A B B	p(A; $[m_p]$)	r(B,A; $[m_{r1}]$)	r(B,A; $[m_{r1}]$)	s(B; $[m_s]$)	$B < A$	$B \leq B$	$m_p m_{r1}^2 m_s$

The set of possible mappings for this case is $\mathcal{M}' = \{\tau_1\}$, therefore $P(\mathcal{M}) = \{\emptyset, \{\tau_1\}\}$. The list of cases is shown in the following table.

Case	Mappings	Formula
C_0	$\neg\tau_1$	$[A \neq B \wedge A > B] \ \wedge \ \neg[(B < A) \wedge B \leq B]$
C_1	$\neg\tau_1$	$[A \neq B \wedge A > B] \ \wedge \ [(B < A) \wedge B \leq B]$

As for the previous database, the unsatisfiability of the formula for case C_0 indicates that Q_2 always obtains the canonical fact, and C_1 produces a satisfiable formula. Thus, τ_1 can always be applied, and the multiplicity of the canonical fact obtained by Q_2 for d_7^0 is $|t_{d_7^0}|_{Q_1(d_7^0)} = m_p m_{r1}^2 m_s$.

d_8^0**:** The only assignment mapping from Q_2 to this database is shown in the following table.

	X Y Z T W	p(X)	r(Y,Z)	r(W,Z)	s(T)	$Y < Z$	$Y \leq W$	$(\|t_{d_8^0}\|_{Q_2(d_8^0)})_{\{\tau_i\}}$
τ_1	A B A A B	p(A; $[m_p]$)	r(B,A; $[m_r]$)	r(B,A; $[m_r]$)	s(A; $[m_s]$)	$B < A$	$B \leq B$	$m_p m_r^2 m_s$

The following list of cases shows that Q_2 always obtains the canonical fact applying the assignment mapping τ_1 (the formula for the case C_0 is again unsatisfiable).

Case	Mappings	Formula
C_0	$\neg\tau_1$	$[A \neq B \wedge A > B] \ \wedge \ \neg[(B < A) \wedge B \leq B]$
C_1	$\neg\tau_1$	$[A \neq B \wedge A > B] \ \wedge \ [(B < A) \wedge B \leq B]$

The multiplicity of the canonical fact obtained by Q_2 for this database is $|t_{d_8^0}|_{Q_1(d_8^0)} = m_p m_r^2 m_s$.

d_{10}^0: There are two assignment mappigns from Q_2 to d_{10}^0:

	X Y Z T W	p(X)	r(Y,Z)	r(W,Z)	s(T)	$Y < Z$	$Y \leq W$	$(\vert t_{d_{10}^0}\vert_{Q_2(d_{10}^0)})_{\{\tau_i\}}$
τ_1	A B A C B	p(A; $[m_p]$)	r(B,A; $[m_{r1}]$)	r(B,A; $[m_{r1}]$)	s(C; $[m_s]$)	$B < A$	$B \leq B$	$m_p m_{r1}^2 m_s$
τ_2	A A C C A	p(A; $[m_p]$)	r(A,C; $[m_{r2}]$)	r(A,C; $[m_{r2}]$)	s(C; $[m_s]$)	$A < C$	$A \leq A$	$m_p m_{r2}^2 m_s$

The list of cases for this database is shown in the following table.

Case	Mappings	Formula
C_0	$\neg\tau_1\neg\tau_2$	$[A \neq B \wedge A \neq C \wedge B \neq C \wedge A > B] \wedge \neg[B < A \wedge B \leq B] \ \wedge \ \neg[A < C \wedge A \leq A]$
C_1	$\neg\tau_1\tau_2$	$[A \neq B \wedge A \neq C \wedge B \neq C \wedge A > B] \wedge \neg[B < A \wedge B \leq B] \ \wedge \ [A < C \wedge A \leq A]$
C_2	$\tau_1\neg\tau_2$	$[A \neq B \wedge A \neq C \wedge B \neq C \wedge A > B] \wedge [B < A \wedge B \leq B] \ \wedge \ \neg[A < C \wedge A \leq A]$
C_3	$\tau_1\tau_2$	$[A \neq B \wedge A \neq C \wedge B \neq C \wedge A > B] \wedge [B < A \wedge B \leq B] \ \wedge \ [A < C \wedge A \leq A]$

Cases C_0 and C_1 produce unsatisfiable formulas, so they are discarded. The formulas generated for the case C_2 (corresponding to the set of mappings $M_2' = \{\tau_1\}$) and C_3(corresponding to the set of mappings $M_3' = \{\tau_1, \tau_2\}$) are satisfiable, therefore Q_2 obtains the canonical fact from d_{10}^0 with the following two multiplicities:

$$(|t_{d_{10}^0}|_{Q_2(d_{10}^0)})_{M_2'} = m_p m_{r1}^2 m_s$$

$$(|t_{d_{10}^0}|_{Q_2(d_{10}^0)})_{M_3'} = m_p m_{r1}^2 m_s + m_p m_{r2}^2 m_s$$

d_{11}^3: There are two assignment mappings from Q_2 to this database.

	X Y Z T W	p(X)	r(Y,Z)	r(W,Z)	s(T)	$Y < Z$	$Y \leq W$	$(\vert t_{d_{11}^3}\vert_{Q_2(d_{11}^3)})_{\{\tau_i\}}$
τ_1	A B A A B	p(A; $[m_p]$)	r(B,A; $[m_{r1}]$)	r(B,A; $[m_{r1}]$)	s(A; $[m_s]$)	$B < A$	$B \leq B$	$m_p m_{r1}^2 m_s$
τ_2	A B A A C	p(A; $[m_p]$)	r(B,A; $[m_{r1}]$)	r(C,A; $[m_{r2}]$)	s(A; $[m_s]$)	$B < A$	$B \leq C$	$m_p m_{r1} m_{r2} m_s$

The following table shows the list of cases:

Case	Mappings	Formula
C_0	$\neg\tau_1\neg\tau_2$	$[A \neq B \wedge A \neq C \wedge B \neq C \wedge A > B \wedge \neg(A > C)] \wedge$ $\neg[B < A \wedge B \leq B] \ \wedge \ \neg[B < A \wedge B \leq C]$
C_1	$\neg\tau_1\tau_2$	$[A \neq B \wedge A \neq C \wedge B \neq C \wedge A > B \wedge \neg(A > C)] \wedge$ $\neg[B < A \wedge B \leq B] \ \wedge \ [B < A \wedge B \leq C]$
C_2	$\tau_1\neg\tau_2$	$[A \neq B \wedge A \neq C \wedge B \neq C \wedge A > B \wedge \neg(A > C)] \wedge$ $[B < A \wedge B \leq B] \ \wedge \ \neg[B < A \wedge B \leq C]$
C_3	$\tau_1\tau_2$	$[A \neq B \wedge A \neq C \wedge B \neq C \wedge A > B \wedge \neg(A > C)] \wedge$ $[B < A \wedge B \leq B] \ \wedge \ [B < A \wedge B \leq C]$

For this database, only case C_3 produces a satisfiable formula. Therefore, Q_2 obtains the canonical fact from d_{11}^3 with the multiplicity:

$$(|t_{d_{11}^3}|_{Q_2(d_{11}^3)})_{M_3'} = m_p m_{r1}^2 m_s + m_p m_{r1} m_{r2} m_s$$

d_{11}^7: There are 4 possible assignment mappings from Q_2 to d_{11}^7:

	X Y Z T W	p(X)	r(Y,Z)	r(W,Z)	s(T)	$Y < Z$	$Y \leq W$	$(\|t_{d_{11}^7}\|_{Q_2(d_{11}^7)})_{\{\tau_i\}}$
τ_1	A B A A B	p(A; $[m_p]$)	r(B,A; $[m_{r1}]$)	r(B,A; $[m_{r1}]$)	s(A; $[m_s]$)	$B < A$	$B \leq B$	$m_p m_{r1}^2 m_s$
τ_2	A B A A C	p(A; $[m_p]$)	r(B,A; $[m_{r1}]$)	r(C,A; $[m_{r2}]$)	s(A; $[m_s]$)	$B < A$	$B \leq C$	$m_p m_{r1} m_{r2} m_s$
τ_3	A C A A B	p(A; $[m_p]$)	r(C,A; $[m_{r2}]$)	r(B,A; $[m_{r1}]$)	s(A; $[m_s]$)	$C < A$	$C \leq B$	$m_p m_{r2} m_{r1} m_s$
τ_4	A C A A C	p(A; $[m_p]$)	r(C,A; $[m_{r2}]$)	r(C,A; $[m_{r2}]$)	s(A; $[m_s]$)	$C < A$	$C \leq C$	$m_p m_{r2}^2 m_s$

For this database, the list of cases consists of $2^4 = 16$ elements, which is the cardinality of $P(\mathcal{M}')$, as the next tables show.

Case	Mappings	Formula
C_0	$\neg\tau_1\neg\tau_2\neg\tau_3\neg\tau_4$	$[A \neq B \wedge A \neq C \wedge B \neq C \wedge \mathbf{A} > \mathbf{B} \wedge A > C] \wedge$ $\neg[\mathbf{B} < \mathbf{A} \wedge B \leq B] \;\wedge\; \neg[B < A \wedge B \leq C] \wedge$ $\neg[C < A \wedge C \leq B] \wedge \neg[C < A \wedge C \leq C]$
C_1	$\neg\tau_1\neg\tau_2\neg\tau_3\tau_4$	$[A \neq B \wedge A \neq C \wedge B \neq C \wedge \mathbf{A} > \mathbf{B} \wedge A > C] \wedge$ $\neg[\mathbf{B} < \mathbf{A} \wedge B \leq B] \;\wedge\; \neg[B < A \wedge B \leq C] \wedge$ $\neg[C < A \wedge C \leq B] \wedge [C < A \wedge C \leq C]$
C_2	$\neg\tau_1\neg\tau_2\tau_3\neg\tau_4$	$[A \neq B \wedge A \neq C \wedge B \neq C \wedge \mathbf{A} > \mathbf{B} \wedge A > C] \wedge$ $\neg[\mathbf{B} < \mathbf{A} \wedge B \leq B] \;\wedge\; \neg[B < A \wedge B \leq C] \wedge$ $[C < A \wedge C \leq B] \wedge \neg[C < A \wedge C \leq C]$
C_3	$\neg\tau_1\tau_2\neg\tau_3\neg\tau_4$	$[A \neq B \wedge A \neq C \wedge B \neq C \wedge \mathbf{A} > \mathbf{B} \wedge A > C] \wedge$ $\neg[\mathbf{B} < \mathbf{A} \wedge B \leq B] \;\wedge\; [B < A \wedge B \leq C] \wedge$ $\neg[C < A \wedge C \leq B] \wedge \neg[C < A \wedge C \leq C]$
C_4	$\tau_1\neg\tau_2\neg\tau_3\neg\tau_4$	$[A \neq B \wedge A \neq C \wedge B \neq C \wedge A > B \wedge \mathbf{A} > \mathbf{C}] \wedge$ $[B < A \wedge B \leq B] \;\wedge\; \neg[B < A \wedge B \leq C] \wedge$ $\neg[C < A \wedge C \leq B] \wedge \neg[\mathbf{C} < \mathbf{A} \wedge C \leq C]$

Case	Mappings	Formula
C_5	$\neg\tau_1\neg\tau_2\tau_3\tau_4$	$[A \neq B \wedge A \neq C \wedge B \neq C \wedge \mathbf{A} > \mathbf{B} \wedge A > C] \wedge$ $\neg[\mathbf{B} < \mathbf{A} \wedge B \leq B] \;\wedge\; \neg[B < A \wedge B \leq C] \wedge$ $[C < A \wedge C \leq B] \wedge [C < A \wedge C \leq C]$
C_6	$\neg\tau_1\tau_2\neg\tau_3\tau_4$	$[A \neq B \wedge A \neq C \wedge B \neq C \wedge \mathbf{A} > \mathbf{B} \wedge A > C] \wedge$ $\neg[\mathbf{B} < \mathbf{A} \wedge B \leq B] \;\wedge\; [B < A \wedge B \leq C] \wedge$ $\neg[C < A \wedge C \leq B] \wedge [C < A \wedge C \leq C]$
C_7	$\tau_1\neg\tau_2\neg\tau_3\tau_4$	$[A \neq B \wedge A \neq C \wedge B \neq C \wedge \mathbf{A} > \mathbf{B} \wedge \mathbf{A} > \mathbf{C}] \wedge$ $[B < A \wedge B \leq B] \;\wedge\; \neg[\mathbf{B} < \mathbf{A} \wedge \mathbf{B} \leq \mathbf{C}] \wedge$ $\neg[\mathbf{C} < \mathbf{A} \wedge \mathbf{C} \leq \mathbf{B}] \wedge [C < A \wedge C \leq C]$
C_8	$\neg\tau_1\tau_2\tau_3\neg\tau_4$	$[A \neq B \wedge A \neq C \wedge B \neq C \wedge \mathbf{A} > \mathbf{B} \wedge A > C] \wedge$ $\neg[\mathbf{B} < \mathbf{A} \wedge B \leq B] \;\wedge\; [B < A \wedge B \leq C] \wedge$ $[C < A \wedge C \leq B] \wedge \neg[C < A \wedge C \leq C]$
C_9	$\tau_1\neg\tau_2\tau_3\neg\tau_4$	$[A \neq B \wedge A \neq C \wedge B \neq C \wedge A > B \wedge \mathbf{A} > \mathbf{C}] \wedge$ $[B < A \wedge B \leq B] \;\wedge\; \neg[B < A \wedge B \leq C] \wedge$ $[C < A \wedge C \leq B] \wedge \neg[\mathbf{C} < \mathbf{A} \wedge C \leq C]$
C_{10}	$\tau_1\tau_2\neg\tau_3\neg\tau_4$	$[A \neq B \wedge A \neq C \wedge B \neq C \wedge A > B \wedge \mathbf{A} > \mathbf{C}] \wedge$ $[B < A \wedge B \leq B] \;\wedge\; [B < A \wedge B \leq C] \wedge$ $\neg[C < A \wedge C \leq B] \wedge \neg[\mathbf{C} < \mathbf{A} \wedge C \leq C]$
C_{11}	$\neg\tau_1\tau_2\tau_3\tau_4$	$[A \neq B \wedge A \neq C \wedge B \neq C \wedge \mathbf{A} > \mathbf{B} \wedge A > C] \wedge$ $\neg[\mathbf{B} < \mathbf{A} \wedge B \leq B] \;\wedge\; [B < A \wedge B \leq C] \wedge$ $[C < A \wedge C \leq B] \wedge [C < A \wedge C \leq C]$
C_{12}	$\tau_1\neg\tau_2\tau_3\tau_4$	$[A \neq B \wedge A \neq C \wedge B \neq C \wedge A > B \wedge A > C] \wedge$ $[B < A \wedge B \leq B] \;\wedge\; \neg[B < A \wedge B \leq C] \wedge$ $[C < A \wedge C \leq B] \wedge [C < A \wedge C \leq C]$
C_{13}	$\tau_1\tau_2\neg\tau_3\tau_4$	$[A \neq B \wedge A \neq C \wedge B \neq C \wedge A > B \wedge A > C] \wedge$ $[B < A \wedge B \leq B] \;\wedge\; [B < A \wedge B \leq C] \wedge$ $\neg[C < A \wedge C \leq B] \wedge [C < A \wedge C \leq C]$
C_{14}	$\tau_1\tau_2\tau_3\neg\tau_4$	$[A \neq B \wedge A \neq C \wedge B \neq C \wedge A > B \wedge A > C] \wedge$ $[B < A \wedge B \leq B] \;\wedge\; [B < A \wedge \mathbf{B} \leq \mathbf{C}] \wedge$ $[C < A \wedge \mathbf{C} \leq \mathbf{B}] \wedge \neg[C < A \wedge C \leq C]$
C_{15}	$\tau_1\tau_2\tau_3\tau_4$	$[A \neq B \wedge A \neq C \wedge B \neq C \wedge A > B \wedge A > C] \wedge$ $[B < A \wedge B \leq B] \;\wedge\; [B < A \wedge \mathbf{B} \leq \mathbf{C}] \wedge$ $[C < A \wedge \mathbf{C} \leq \mathbf{B}] \wedge [C < A \wedge C \leq C]$

Testing the satisfiability of these 15 formulas (the method presented in Section 6.5 can be used to test it, but the inequalities that will produce the unsatisfiability of the formulas are shown in boldface in the table), only two of them are satisfiable: those corresponding to cases C_{12} and C_{13}. Therefore, the multiplicities of the canonical fact obtained by Q_2 are the following:

$$(|t_{d_{11}^7}|_{Q_2(d_{11}^7)})_{M'_{12}} = m_p m_{r1}^2 m_s + m_p m_{r2} m_{r1} m_s + m_p m_{r2}^2 m_s$$

$$(|t_{d_{11}^7}|_{Q_2(d_{11}^7)})_{M'_{13}} = m_p m_{r1}^2 m_s + m_p m_{r1} m_{r2} m_s + m_p m_{r2}^2 m_s$$

d_{12}^0: The two possible assignment mappings from Q_2 to this database are shown in the following table.

	X Y Z T W	p(X)	r(Y,Z)	r(W,Z)	s(T)	$Y < Z$	$Y \leq W$	$(\|t_{d_{12}^0}\|_{Q_2(d_{12}^0)})_{\{\tau_i\}}$
τ_1	B A B C A	p(B; $[m_p]$)	r(A,B; $[m_{r1}]$)	r(A,B; $[m_{r1}]$)	s(C; $[m_s]$)	$A < B$	$A \leq A$	$m_p m_{r1}^2 m_s$
τ_2	B A C C A	p(B; $[m_p]$)	r(A,C; $[m_{r2}]$)	r(A,C; $[m_{r2}]$)	s(C; $[m_s]$)	$A < C$	$A \leq A$	$m_p m_{r2}^2 m_s$

The following table shows the list of 4 cases that correspond to all possible sets of assignment mappings from Q_2 to d_{12}^0.

Case	Mappings	Formula
C_0	$\neg\tau_1\neg\tau_2$	$[A \neq B \wedge A \neq C \wedge B \neq C \wedge B > A] \wedge$ $\neg[A < B \wedge A \leq A] \wedge \neg[A < C \wedge A \leq A]$
C_1	$\neg\tau_1\tau_2$	$[A \neq B \wedge A \neq C \wedge B \neq C \wedge B > A] \wedge$ $\neg[A < B \wedge A \leq A] \wedge [A < C \wedge A \leq A]$
C_2	$\tau_1\neg\tau_2$	$[A \neq B \wedge A \neq C \wedge B \neq C \wedge B > A] \wedge$ $[A < B \wedge A \leq A] \wedge \neg[A < C \wedge A \leq A]$
C_3	$\tau_1\tau_2$	$[A \neq B \wedge A \neq C \wedge B \neq C \wedge B > A] \wedge$ $[A < B \wedge A \leq A] \wedge [A < C \wedge A \leq A]$

Only C_2 and C_3 produce satisfiable formulas, so Q_2 can derive the canonical fact from d_{12}^0 with the following two multiplicities:

$$(|t_{d_{12}^0}|_{Q_2(d_{12}^0)})_{M'_2} = m_p m_{r1}^2 m_s$$

$$(|t_{d_{12}^0}|_{Q_2(d_{12}^0)})_{M'_3} = m_p m_{r1}^2 m_s + m_p m_{r2}^2 m_s$$

d_{13}^0: There are the following two assignment mappings from Q_2 to this database.

	X Y Z T W	p(X)	r(Y,Z)	r(W,Z)	s(T)	$Y < Z$	$Y \leq W$	$(\|t_{d_{13}^0}\|_{Q_2(d_{13}^0)})_{\{\tau_i\}}$
τ_1	B A B A A	p(B; $[m_p]$)	r(A,B; $[m_{r1}]$)	r(A,B; $[m_{r1}]$)	s(A; $[m_s]$)	$A < B$	$A \leq A$	$m_p m_{r1}^2 m_s$
τ_2	B C A A C	p(B; $[m_p]$)	r(C,A; $[m_{r2}]$)	r(C,A; $[m_{r2}]$)	s(A; $[m_s]$)	$C < A$	$C \leq C$	$m_p m_{r2}^2 m_s$

The list of cases is shown in the following table, where only cases C_2 and C_3 produce satisfiable formulas.

Case	Mappings	Formula
C_0	$\neg\tau_1\neg\tau_2$	$[A \neq B \wedge A \neq C \wedge B \neq C \wedge B > A] \wedge$ $\neg[A < B \wedge A \leq A] \wedge \neg[C < A \wedge C \leq C]$
C_1	$\neg\tau_1\tau_2$	$[A \neq B \wedge A \neq C \wedge B \neq C \wedge B > A] \wedge$ $\neg[A < B \wedge A \leq A] \wedge [C < A \wedge C \leq C]$
C_2	$\tau_1\neg\tau_2$	$[A \neq B \wedge A \neq C \wedge B \neq C \wedge B > A] \wedge$ $[A < B \wedge A \leq A] \wedge \neg[C < A \wedge C \leq C]$
C_3	$\tau_1\tau_2$	$[A \neq B \wedge A \neq C \wedge B \neq C \wedge B > A] \wedge$ $[A < B \wedge A \leq A] \wedge [C < A \wedge C \leq C]$

The multiplicities of the canonical fact obtained by Q_2 from this database are:

$$(|t_{d^0_{13}}|_{Q_2(d^0_{13})})_{M'_2} = m_p m^2_{r1} m_s$$

$$(|t_{d^0_{13}}|_{Q_2(d^0_{13})})_{M'_3} = m_p m^2_{r1} m_s + m_p m^2_{r2} m_s$$

d^0_{14}: The following table shows the unique assignment mapping from Q_2 to this database.

	X Y Z T W	p(X)	r(Y,Z)	r(W,Z)	s(T)	$Y < Z$	$Y \leq W$	$(\lvert t_{d^0_{14}}\rvert_{Q_2(d^0_{14})})_{\{\tau_i\}}$
τ_1	B C B A C	p(B; $[m_p]$)	r(C,B; $[m_{r1}]$)	r(C,B; $[m_{r1}]$)	s(A; $[m_s]$)	$C < B$	$C \leq C$	$m_p m^2_{r1} m_s$

Case	Mappings	Formula
C_0	$\neg\tau_1\neg\tau_2$	$[A \neq B \wedge A \neq C \wedge B \neq C \wedge B > C] \wedge \neg[C < B \wedge C \leq C]]$
C_1	$\neg\tau_1\neg\tau_2$	$[A \neq B \wedge A \neq C \wedge B \neq C \wedge B > C] \wedge [C < B \wedge C \leq C]$

Since the formula for the case C_0 is unsatisfiable, Q_2 always derives the canonical fact from this database, with the following multiplicity:

$$(|t_{d^0_{14}}|_{Q_2(d^0_{14})})_{M'_1} = m_p m^2_{r1} m_s$$

d^0_{15}: There are 2 assignment mappings from Q_2 to this database.

	X Y Z T W	p(X)	r(Y,Z)	r(W,Z)	s(T)	$Y < Z$	$Y \leq W$	$(\lvert t_{d^0_{15}}\rvert_{Q_2(d^0_{15})})_{\{\tau_i\}}$
τ_1	A B A D B	p(A; $[m_p]$)	r(B,A; $[m_{r1}]$)	r(B,A; $[m_{r1}]$)	s(D; $[m_s]$)	$B < A$	$B \leq B$	$m_p m^2_{r1} m_s$
τ_1	A C D D D	p(A; $[m_p]$)	r(C,D; $[m_{r2}]$)	r(C,D; $[m_{r2}]$)	s(D; $[m_s]$)	$C < D$	$C \leq C$	$m_p m^2_{r2} m_s$

The list of the 4 possible sets of assignment mappings from Q_2 to d^0_{15} are shown in the following table.

Case	Mappings	Formula
C_0	$\neg\tau_1\neg\tau_2$	$[A \neq B \wedge A \neq C \wedge A \neq D \wedge B \neq C \wedge B \neq D \wedge C \neq D \wedge A > B] \wedge$ $\neg[B < A \wedge B \leq B] \wedge \neg[C < D \wedge C \leq C]$
C_1	$\neg\tau_1\tau_2$	$[A \neq B \wedge A \neq C \wedge A \neq D \wedge B \neq C \wedge B \neq D \wedge C \neq D \wedge A > B] \wedge$ $\neg[B < A \wedge B \leq B] \wedge [C < D \wedge C \leq C]$
C_2	$\tau_1\neg\tau_2$	$[A \neq B \wedge A \neq C \wedge A \neq D \wedge B \neq C \wedge B \neq D \wedge C \neq D \wedge A > B] \wedge$ $[B < A \wedge B \leq B] \wedge \neg[C < D \wedge C \leq C]$
C_3	$\tau_1\tau_2$	$[A \neq B \wedge A \neq C \wedge A \neq D \wedge B \neq C \wedge B \neq D \wedge C \neq D \wedge A > B] \wedge$ $[B < A \wedge B \leq B] \wedge [C < D \wedge C \leq C]$

Again, only cases C_2 and C_3 produce satisfiable formulas. Thus, the multiplicities of the canonical fact obtained by Q_2 from d^0_{15} are:

$$(|t_{d^0_{15}}|_{Q_2(d^0_{15})})_{M'_2} = m_p m^2_{r1} m_s$$

$$(|t_{d^0_{15}}|_{Q_2(d^0_{15})})_{M'_3} = m_p m^2_{r1} m_s + m_p m^2_{r2} m_s$$

Test the bag containment:

At this point, the multiplicities of the canonical fact obtained from all canonical databases by either Q_1 or Q_2 are known, and must be compared in order to test the bag containment. The following table summarizes all these multiplicities. Note that the column that shows the multiplicity of the canonical fact obtained by the application of Q_2 to any canonical database can have several rows for the same database, when Q_2 derives the canonical fact with different multiplicities (for example, for database d^7_{11}).

CDB	$\lvert t_d\rvert_{Q_1(d)}$	$\lvert t_d\rvert_{Q_2(d)}$
d^1_4	$m_p m_{r2} m_{r1} m_s + m_p m^2_{r1} m_s$	$m_p m^2_{r1} m_s + m_p m_{r1} m_{r2} m_s$
d^0_5	$m_p m_{r1} m_{r2} m_s$	$m_p m^2_{r1} m_s$
d^0_7	$m_p m_{r1} m_{r2} m_s$	$m_p m^2_{r1} m_s$
d^0_8	$m_p m_{r1} m_{r2} m_s$	$m_p m^2_{r} m_s$
d^0_{10}	$m_p m_{r1} m_{r2} m_s$	$m_p m^2_{r1} m_s$
		$m_p m^2_{r1} m_s + m_p m^2_{r2} m_s$
d^3_{11}	$m_p m_{r1} m_{r2} m_s + m_p m^2_{r1} m_s$	$m_p m^2_{r1} m_s$
d^7_{11}	$m_p m_{r1} m_{r2} m_s + m_p m^2_{r1} m_s + m_p m^2_{r2} m_s + m_p m_{r2} m_{r1} m_s$	$m_p m^2_{r1} m_s + m_p m_{r2} m_{r1} m_s + m_p m^2_{r2} m_s$
		$m_p m^2_{r1} m_s + m_p m_{r1} m_{r2} m_s + m_p m^2_{r2} m_s$
d^0_{12}	$m_p m_{r1} m_{r2} m_s$	$m_p m^2_{r1} m_s$
		$m_p m^2_{r1} m_s + m_p m^2_{r2} m_s$
d^0_{13}	$m_p m_{r1} m_{r2} m_s$	$m_p m^2_{r1} m_s$
		$m_p m^2_{r1} m_s + m_p m^2_{r2} m_s$
d^0_{14}	$m_p m_{r1} m_{r2} m_s$	$m_p m^2_{r1} m_s$
d^0_{15}	$m_p m_{r1} m_{r2} m_s$	$m_p m^2_{r1} m_s$
		$m_p m^2_{r1} m_s + m_p m^2_{r2} m_s$

The procedure must now compare the multiplicities of the canonical fact obtained by Q_1 and Q_2, represented by polynomials. It is easy to test that the multiplicity of $t_{d^0_5}$ obtained by Q_2 is not greater than (or equal to) the multiplcity obtained by Q_1 (for example, if $m_p = m_{r1} = m_s = 1$ and $m_{r2} = 2$, the multiplicity obtained by Q_1 would be greater). Therefore, the procedure concludes that $Q_1 \not\leq_b Q_2$.

Another canonical database that shows that the containment does not hold is d^0_{13}: The multiplicity obtained by Q_1 is:

$$|t_{d^0_{13}}|_{Q_1(d^0_{13})} = m_p m_{r1} m_{r2} m_s$$

And the multiplicities obtained by Q_2 are:

$$(|t_{d^0_{13}}|_{Q_2(d^0_{13})})_{M'_2} = m_p m^2_{r1} m_s$$

$$(|t_{d^0_{13}}|_{Q_2(d^0_{13})})_{M'_3} = m_p m^2_{r1} m_s + m_p m^2_{r2} m_s$$

Since $(|t_{d^0_{13}}|_{Q_2(d^0_{13})})_{M'_2}$ is not at least as high as $|t_{d^0_{13}}|_{Q_1(d^0_{13})}$ (even when $(|t_{d^0_{13}}|_{Q_2(d^0_{13})})_{M'_3}$ is), that means that the containment does not hold. □

9.7 Summary

This chapter has described another important contribution of this Thesis, the application of *QCC* to test bag containment of inequality queries, because it solves a problem for which there were no results so far. It combines the multiplicities in the facts of the canonical databases (as used to test bag containment of equality queries) and constraints associated to each canonical database (as used to test set containment of inequality queries) to offer a procedure to test bag containment of inequality queries. This problem is reduced, therefore, to the problems of checking the unsatisfiability of several formulas, and the comparison of pairs of polynomials over $\mathbb{Z}^+$.

Chapter 10

Conclusions and future work

In this Thesis, we have studied the problem of containment of conjunctive queries under four perspectives, taking into account the presence or absence of built-in predicates in the conjunctive queries, and the underlying (set or bag) semantics. The four perspectives generated using these two factors lead to four types of conjunctive query containment:

- Set containment of equality queries;
- Bag containment of equality queries;
- Set containment of inequality queries; and
- Bag containment of inequality queries.

We first define the set and bag semantics and how to apply a query to a database under both of them. Next, we review the state of the art in these four types of containment, showing that the results achieved so far are not applicable for the problem of testing the set and bag containment of inequality queries.

The main contribution of this Thesis is QCC (Query Containment Checker), a general procedure that can be used to check the previous four types of conjunctive query containment. The key concept of QCC is the use of the *canonical database set* built from the predicates in the body of a query Q_1 ($CDBS(Q_1)$), because it allows us to test the containment of conjunctive queries using only a finite (usually small) set of databases, those in $CDBS(Q_1)$, instead of using an infinite number of databases (as described in the formal definition of query containment).

QCC is a procedure that is capable of testing conjunctive query containment for those cases already solved, such as set containment of equality queries [CM77] or bag containment of equality queries [CV93, BH97], but it is also capable of successfully performing these tests for other types of query containment problems (set and bag containment of inequality queries) for which there were no results so far. It is our belief that QCC can also be used to test other types of query containment, such us containment of queries with negated subgoals, or containment of nonrecursive Datalog programs. Our current research is directed at finding a way to apply QCC to test these containments.

Bibliography

[AHV95] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of databases.* Adison-Wesley, 1995.

[ASU79a] A. V. Aho, Yehoshua Sagiv, and Jeffrey D. Ullman. Efficient optimization of a class of relational queries. *ACM Transactions on Database Systems*, 4(4):435–454, 1979.

[ASU79b] A. V. Aho, Yehoshua Sagiv, and Jeffrey D. Ullman. Equivalence of relational expressions. *SIAM J. of Computing*, 8(2):218–246, 1979.

[BH97] Nieves R. Brisaboa and Héctor J. Hernández. Testing bag-containment of conjunctive queries. *Acta Informatica*, 34:557–578, 1997.

[BHPP98] 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. Second East Sympsium (ADBIS'98)*, number 1475 in Lecture Notes in Computer Science, pages 46–57, Poznan, Poland, September 1998. Springer-Verlag.

[Bri97] 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.

[CGT89] S. Ceri, G. Gottlob, and L. Tanka. What you Always Wanted to Know about Datalog (and Never Dared to Ask). *IEEE Transactions on Knowledge and Data Engineering*, 1(1):146–166, March 1989.

[CM77] 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.

[Cod70] E. F. Codd. A relational model for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

[CV93] 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.

[DGK82] U. Dayal, N. Goodman, and R. H. Katz. An extended relational algebra with control over duplicate elimination. In *Proc. First ACM Symposium on Principles of Database Systems*, pages 117–123, 1982.

[Fag93] R. Fagin. Finite model theory – a personal perspective. *Theoretical Computer Science*, 116(1):3–31, August 1993.

[GSW96] Sha Guo, Wei Sun, and Mark A. Weiss. Solving satisfiability and implication problems in database systems. *ACM Transactions on Database Systems*, 21(2):270–293, 1996.

[IO97] Naci S. Ishakbeyoglu and Z. Meral Ozsoyoglu. Testing satisfiability of a conjunction of inequalities. In *International Symposium on Computer and Information Systems (ISCIS XII)*, pages 148–154, Anlatya, Turkey, October 1997.

[IR92] Yannis E. Ioannidis and Raghu Ramakrishnan. Generalized containment of conjunctive queries. Technical report, Computer Science Department. University of Wisconsin, Madison, WI 53706, January 1992.

[IR94] Yannis E. Ioannidis and Raghu Ramakrishnan. Containment of conjunctive queries beyond relations as sets. Technical report, Computer Science Department. University of Wisconsin, Madison, WI, 1994.

[IS97] 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.

[Kla86] A. Klausner. *Multirelations in Relational Databases.* PhD thesis, Harvard University, 1986.

[Klu88] Anthony Klug. On conjunctive queries containing inequalities. *Journal of the ACM*, 35(1):146–160, 1988.

[Llo87] J. W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, second, extended edition, 1987.

[Sol79] M. K. Solomon. Some properties of relational expressions. In *Proceedings of the ACM Southeast Regional Conference*, pages 111–116, ACM, New Yotk, 1979.

[Tra50] B. A. Trakhtenbrot. The imposibility of an algorithm for the decision problem for finite models. *Doklady Akademii Naurk SSR*, 70:569–572, 1950.

[Ull82] Jeffrey D. Ullman. *Principles of Database Systems.* Computer Science Press, second edition, 1982.

[Ull89] Jeffrey D. Ullman. *Principles of Database and Knowledge-base Systems*, volume 1 and 2. Computer Science Press, 1988-1989.

[vdM97] 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.

[ZO93] X. Zhang and Z. Meral Ozsoyoglu. On efficient reasoning with implication constraints. In *Proc. of 3rd International Conference on Deductive and Object-Oriented Databases (DOOD'93)*, pages 236–252, Phoenix, Arizona, December 1993.