Containment of conjunctive queries with built-in predicates with variables and constants over any ordered domain *

Nieves R. Brisaboa[1], Héctor J. Hernández[2], José R. Paramá[1], and Miguel R. Penabad[1]

[1] Departamento de Computación, Universidade da Coruña, 15071 A Coruña, Spain.
{brisaboa,parama,penabad}@udc.es
[2] Laboratory for Logic, Databases, and Advanced Programming, New Mexico State University, Las Cruces, NM 88003-8001, USA.
hector@cs.nmsu.edu

Abstract. In this paper, we consider conjunctive queries with built-in predicates of the form $X < Y$, $X \leq Y$, $X = Y$, or $X \neq Y$, where X and Y are variables or constants from a totally ordered domain. We present a sufficient and necessary condition to test containment among these kinds of queries. Klug [8] left the problem open for the case when the domain is nondense, like the integers. Ullman [11] gave only a sufficient condition for the containment of conjunctive queries with built-in predicates and integer variables. Our test is based in a method that uses a new idea: the representation of an infinite number of databases by a finite set of, what we call, canonical databases, that use variables that denote uninterpreted constants.

1 Introduction

A *conjunctive query* is a safe, nonrecursive datalog rule [11] (with subgoals defined exclusively on extensional database predicates); it is equivalent to a relational algebra expression that uses the selection, projection, and Cartesian product operators. A conjunctive query Q is contained in a conjunctive query Q', denoted $Q \subseteq Q'$, if for any input database D, $Q(D)$, the output of Q when D is its input, is contained in $Q'(D)$.

Testing equivalence of conjunctive queries is a basic problem that query optimizers must be able to solve. Their main goal of finding a plan to execute a query as efficiently as possible could benefit from the ability to chose between two equivalent queries, in the very likely case that one of them is more expensive to process than the other one. Since checking equivalence of two queries can be done by testing the mutual containment of the queries involved, this latter problem has been studied. Because of its importance, the problem of testing

* This work was partially supported by CICYT grant TEL96-1390-C02 and by NSF grant HRD-9628450.

containment of two conjunctive queries has been studied extensively, and it is well understood [1, 2, 5]. In particular, there is an exact condition to test it.

However, testing containment of conjunctive queries with built-in predicates is a problem that has not been fully solved. Klug [8] solves it for totally ordered dense domains, but not for nondense domains such as the integers. Ibarra and Su [7] show that this problem is decidable for linear constraints, but no effective procedure to test it is given. For built-in predicates of the form $X < Y$, $X \leq Y$, $X = Y$, or $X \neq Y$ over a nondense domain, we only know a sufficient condition [11]. The following example, adapted from Example 14.7 from [11], shows where that condition fails to be necessary[1].

Example 1. Let Q_1 and Q_2 be the following conjunctive queries.
$Q_1 : p(X, Y) : -q(X, Y), r(U, V), r(V, U), X \geq Y.$
$Q_2 : p(X, Y) : -q(X, Y), r(U, V), U \leq V.$

We want to know if $Q_1 \subseteq Q_2$. The test in [11] requires a containment mapping from Q_2 to Q_1 and that the built-in predicates of Q_2 be implied by the ones in Q_1. For this couple of queries, there are two ways to map the ordinary subgoals of Q_2 to the ones in Q_1: $h_1(X) = h_2(X) = X$, $h_1(Y) = h_2(Y) = Y$, $h_1(U) = h_2(V) = U$, $h_1(V) = h_2(U) = V$. Under any of the two mappings, $h_i(U \leq V)$ is not implied by $X \geq Y$, the built-in predicate of Q_1, and thence that test cannot tell whether the containment holds even though Q_1 is contained in Q_2. □

In [4] we showed with success that the idea of *canonical databases* can be used in testing containment of conjunctive queries under bag-semantics. The canonical database set built form the body of a query Q, $CDBS(Q)$, is a finite set of databases, whose tuples contain *uninterpreted constants* that represent in a finite way the infinite number of possible databases that one has to consider in a solution to containment problems. In this paper we use those kinds of databases to present a necessary and sufficient condition to test containment of two conjunctive queries with built-in predicates. Our condition solves the open problem [8] of finding an exact test for nondense domains like the integers.

The plan for the rest of the paper is as follows. In Section 2, we give some definitions. In Section 3, we define canonical databases. In Section 4 we present our condition together with an example of its application. The last section contains our conclusions.

2 Definitions and Notation

We assume the definitions in [11] and only give some nonstandard definitions.

A *conjunctive query* is a safe, nonrecursive datalog rule [11]. We consider conjunctive queries with built-in predicates of the form $X < Y$, $X \leq Y$, $X = Y$, and $X \neq Y$, where X, Y, and Z are variables or constants from a totally ordered domain.

[1] The conditions required to have an exact characterization of the containment among two conjunctive queries with built-in predicates are so subtle that in Theorem 8.1 of [10] a wrong necessary condition is given.

A *database schema* is a finite set of predicate names. We assume that all predicates used in this paper are implicitly in $\mathcal{U}$, a fixed database schema, and that there is a set of constants called the *Herbrand universe*, which in this paper is any totally ordered domain. The *Herbrand base* $B_{\mathcal{U}}$ for $\mathcal{U}$ is the set of all ground atoms that can be formed by using predicate names in $\mathcal{U}$ and constants from the Herbrand universe [9].

A *database* is a finite subset of the Herbrand base. In the examples, we shall represent databases as tables showing the tuples defined on each predicate.

Two databases are *isomorphic* if they are identical up to a consistent renaming of constants. For example, the databases $D = \{\ r(a,b),\ p(b,a),\ r(b,b)\ \}$ and $D' = \{\ r(1,2),\ p(2,1),\ r(2,2)\ \}$ are isomorphic.

The semantics for conjunctive queries is defined in terms of assignment mappings [6]. An *assignment mapping* of a conjunctive query Q into a database D is an assignment of the constants in the facts in D to the variables of Q such that every atom in the body of Q is mapped to a fact that is in D.

Example 2. Let $Q = q(X,Y) :- r(X,U), r(V,U), r(Y,U)$, let $D = \{\ r(a,b),\ r(c,b)\}$, and let α be the following assignment mapping of Q into D: $\alpha(X) = a$, $\alpha(U) = b$, $\alpha(V) = a$, $\alpha(Y) = c$. Then $q(a,c) \in Q(D)$, since α maps $r(X,U)$ and $r(V,U)$ to $r(a,b)$, and it maps $r(Y,U)$ to $r(c,b)$. □

3 Canonical Databases

In this section we introduce canonical databases, which are a finite representation of the infinite number of databases that can be the input to a conjunctive query. In the rest of this section, we will show an example of the canonical databases set for a query Q, denoted by $CDBS(Q)$; an algorithm that builds $CDBS(Q)$ for any conjunctive query Q; and how to apply a conjunctive query to a canonical database

3.1 Example of Building $CDBS(Q)$

The objective of building the set of canonical databases for a specific query Q is to cover all the possible mappings between that query and any database. The next example illustrates this. After the example, we give the formal definition of $CDBS(Q)$.

Example 3. Let us consider the query $Q = q(X,Y,Z) :- r(X,U),\ r(U,Z),\ p(U,Y),\ X > Y$. We shall use the atoms in the body of Q to build $CDBS(Q)$. To do that it is necessary to consider all the different possible mappings of variables in Q to any database. We use the letters A, B, C, D (which represent uninterpreted constants) to identify the values to which the variables in Q could be mapped.

The following cases list the different possible mappings; each case shows several d_i's, where each d_i represents an element of $CDBS(Q)$. Each canonical

database d_i has attached two sets of constraints. The first set contains constraints to enforce that each uninterpreted constant is different of any other in d_i[2]. The second set contains a constraint for each built-in predicate in Q, but the variables in the built-in predicates are transformed in the corresponding uninterpreted constants. These constraints will be denoted by $constraints(d_i)$; for d_1, $constraints(d_1) = (A \neq B \wedge A \neq C \wedge A \neq D \wedge B \neq C \wedge B \neq D \wedge C \neq D) \wedge (\theta_1(X) > \theta_1(Y))$; for d_2, $constraints(d_2) = A \neq B \wedge (\theta_2(X) = A) > (\theta_2(Y) = A)$, which means that such a database is not consistent with the constraints; d_3, d_6, d_9, and d_{15} are also not consistent with their constraints and, hence, it is not possible to build an extensional database isomorphic to them.

Case 1: Each variable in Q is mapped to a different value. Then a canonical database, like the one shown on Table 1, is generated.

Table 1. Canonical Database for Case 1 of Example 3.1

NAME	CDB		MAPPING	$constraints(d_1)$
	r	p		$A > B \wedge$
d_1	(A, D)	(D, B)	$\theta_1(X) = A;\ \ \theta_1(Y) = B;$	$A \neq B \wedge\ A \neq C \wedge\ A \neq D \wedge$
	(D, C)		$\theta_1(Z) = C;\ \ \theta_1(U) = D$	$B \neq C \wedge\ B \neq D \wedge\ C \neq D$

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

Table 2. Canonical Databases for Case 2 of Example 3.1

NAME	CDB		MAPPING	$constraints(d_i)$
	r	p		
d_2	(A,B)	(B,A)	$\theta_2(X) = \theta_2(Y) = \theta_2(Z) = A;\ \ \theta_2(U) = B$	$A > A \wedge A \neq B$
	(B,A)			d_2 is not consistent
d_3	(A,A)	(A,A)	$\theta_3(X) = \theta_3(Y) = \theta_3(U) = A;\ \ \theta_3(Z) = B$	$A > A \wedge A \neq B$
	(A,B)			d_3 is not consistent
d_4	(A, A)	(A, B)	$\theta_4(X) = \theta_4(Z) = \theta_4(U) = A;\ \ \theta_4(Y) = B$	$A > B \wedge A \neq B$
d_5	(B, A)	(A, A)	$\theta_5(Y) = \theta_5(Z) = \theta_5(U) = A;\ \ \theta_5(X) = B$	$B > A \wedge A \neq B$
	(A, A)			

Case 3: Two variables are mapped to the same value and the other two are mapped to another value. Table 3 shows the three canonical databases generated in this case.

[2] There are canonical databases for each of the possible equalities among the variables in Q.

Table 3. Canonical Databases for Case 3 of Example 3.1

NAME	CDB		MAPPING	$constraints(d_i)$
	r	p		
d_6	(A,B) (B,B)	(B,A)	$\theta_6(X) = \theta_6(Y) = A;\quad \theta_6(Z) = \theta_6(U) = B$	$A > A \wedge A \neq B$ d_6 is not consistent
d_7	(A, B) (B, A)	(B, B)	$\theta_7(X) = \theta_7(Z) = A;\quad \theta_7(Y) = \theta_7(U) = B$	$A > B \wedge A \neq B$
d_8	(A, A) (A, B)	(A, B)	$\theta_8(X) = \theta_8(U) = A;\quad \theta_8(Y) = \theta_8(Z) = B$	$A > B \wedge A \neq B$

Case 4: Two variables are mapped to the same value and the other two are mapped to different values. The canonical databases generated in this case are shown on Table 4.

Table 4. Canonical Databases for Case 4 of Example 3.1

NAME	CDB		MAPPING	$constraints(d_i)$
	r	p		
d_9	(A,C) (C,B)	(C,A)	$\theta_9(X) = \theta_9(Y) = A;$ $\theta_9(Z) = B;\quad \theta_9(U) = C$	$A > A \wedge$ $A \neq B \wedge A \neq C \wedge B \neq C$ d_9 is not consistent
d_{10}	(A, C) (C, A)	(C, B)	$\theta_{10}(X) = \theta_{10}(Z) = A;$ $\theta_{10}(Y) = B;\quad \theta_{10}(U) = C$	$A > B \wedge$ $A \neq B \wedge A \neq C \wedge B \neq C$
d_{11}	(A, A) (A, C)	(A, B)	$\theta_{11}(X) = \theta_{11}(U) = A;$ $\theta_{11}(Y) = B;\quad \theta_{11}(Z) = C$	$A > B \wedge$ $A \neq B \wedge A \neq C \wedge B \neq C$
d_{12}	(B, C) (C, A)	(C, A)	$\theta_{12}(Y) = \theta_{12}(Z) = A;$ $\theta_{12}(X) = B;\quad \theta_{12}(U) = C$	$B > A \wedge$ $A \neq B \wedge A \neq C \wedge B \neq C$
d_{13}	(B, A) (A, C)	(A, A)	$\theta_{13}(Y) = \theta_{13}(U) = A;$ $\theta_{13}(X) = B;\quad \theta_{13}(Z) = C$	$B > A \wedge$ $A \neq B \wedge A \neq C \wedge B \neq C$
d_{14}	(B, A) (A, A)	(A, C)	$\theta_{14}(Z) = \theta_{14}(U) = A;$ $\theta_{14}(X) = B;\quad \theta_{14}(Y) = C$	$B > C \wedge$ $A \neq B \wedge A \neq C \wedge B \neq C$

Case 5: The four variables are mapped to the same value. Table 5 shows the canonical database generated in this case.

Thus there are 15 canonical databases for Q, but not all of them are consistent with their constraints. Notice that not all the possible canonical databases are required. For example, the mapping of X and Y to B and U and Z to D would produce a canonical database that has the same equalities as d_6 (it would be isomorphic to d_6). With four variables there are 256 different possible canonical databases, but there are only 15 different (nonisomorphic) ones. In [3], the reader

Table 5. Canonical Database for Case 5 of Example 3.1

NAME	CDB		MAPPING	$constraints(d_i)$
	r	p		
d_{15}	(A,A)	(A,A)	$\theta_{15}(X) = \theta_{15}(U) = \theta_{15}(Y) = \theta_{15}(Z) = A$	$A > A$ d_{15} is not consistent

can find a procedure to compute the number of canonical databases in terms of the number of variables in the conjunctive query. □

3.2 Algorithm to build $CDBS(Q)$

Definition of Canonical Databases

Before describing the algorithm it is necessary to define some concepts.
Let Q be a conjunctive query. Then, $db(Q)$ is the set of atoms

$$db(Q) = \{p_i(V_1, \cdots, V_{k_i}) \mid p_i(V_1, \cdots, V_{k_i}) \text{ is an ordinary atom in the body of } Q\}$$

We shall use $db(Q)$ to construct $CDBS(Q)$. The idea is to define below a set of mappings that will be applied to $db(Q)$: for each mapping that we define, an element of $CDBS(Q)$ will be generated along with the restriction that all its variables are distinct. The mappings will be defined on the set of variables found in the atoms in $db(Q)$. We define these next.

Let $V_Q = \langle V_1, \cdots, V_q \rangle$ be an ordering of all the variables that appear in the atoms in $db(Q)$. Let $A_Q = \{A_1, \cdots, A_q\}$ be q new, distinct identifiers ($A_i \neq A_j$ if $i \neq j, 1 \leq i, j \leq q$); they represent uninterpreted constants. We now define the mappings mentioned above, called Q-mappings, from V_Q to A_Q. A *Q-mapping* θ is a q-tuple $\theta = (A_{i_1}, ..., A_{i_q})$ where $1 \leq i_1, i_2, ..., i_q \leq q$. This tuple denotes the mapping $\theta(V_1) = A_{i_1}, ..., \theta(V_q) = A_{i_q}$. Thus Q-mappings are shorthands for the mappings we need to use to produce the canonical databases.

These Q-mappings can be applied to an atom or to $db(Q)$. Let θ be a Q-mapping. Let $p_i(U_1, ..., U_{r_i})$ be an atom. The application of θ to $p_i(U_1, ..., U_{r_i})$ is defined as $\theta(p_i(U_1, ..., U_{r_i})) = p_i(\theta(U_1), ..., \theta(U_{r_i}))$. The application of θ to $db(Q)$ is defined as the canonical database $\theta(db(Q)) = \{p_i(\theta(U_1),\ldots, \theta(U_{r_i}))$ $|p_i(U_1,\ldots, U_{r_i}) \in db(Q)\}$.

The following example demonstrates these definitions.

Example 4. Let us consider the query $Q = q(X, Y, Z) : -r(X, U),\ r(U, Z),\ p(U, Y),\ X \geq Y$. Then $V_Q = \langle X, Y, Z, U \rangle$ is an ordering of the variables in Q; and $A_Q = \{A_1, A_2, A_3, A_4\}$; $db(Q)$ in this example is $\{r(X, U), r(U, Z), p(U, Y)\}$. Let the Q-mapping θ be (A_4, A_3, A_4, A_4); θ denotes the mapping $\theta(X) = \theta(Z) = \theta(U) = A_4$ and $\theta(Y) = A_3$. Then $\theta(db(Q)) = \{r(A_4, A_4), p(A_4, A_3)\}$; with this database the constraints $A_3 \neq A_4$ and $A_3 \leq A_4$ will be associated. We will show later how to deal with this pairs of constraints. □

Isomorphic Q-mappings, which are used to define a minimal number of canonical databases, are defined as follows. Two Q-mappings $\theta_1 = (A_{i_1}, ..., A_{i_q})$ and $\theta_2 = (A_{j_1}, ..., A_{j_q})$ are *isomorphic* if there are two mappings γ_1 and γ_2 (from A_Q to A_Q) such that $(\gamma_1(A_{i_1}), ..., \gamma_1(A_{i_q})) = \theta_2$, and $(\gamma_2(A_{j_1}), ..., \gamma_2(A_{j_q})) = \theta_1$.

Example 5. The Q-mapping $\beta = (A_3, A_3, A_1, A_3)$ is isomorphic to $\alpha = (A_4, A_4, A_3, A_4)$; and (A_1, A_1, A_1, A_1) is isomorphic to (A_2, A_2, A_2, A_2). □

We now define some concepts associated to the canonical databases.

Let Q_1 be the following conjunctive query. $Q_1 : I : -J_1, \cdots, J_q, K_1, \cdots, K_n$, where the K_i's are the built-in predicates of Q_1. Then for all d in $CDBS(Q_1)$, we define:

1. θ_d to be the Q-mapping that we used to build d (from Q_1);
2. t_d to be the fact $\theta_d(I)$;
3. $constraints(d)$ to be the constraints that d has to satisfy, i.e: $\theta_d(K_1 \wedge \cdots \wedge K_n) \wedge (A_i \neq A_j, \forall i, j, 1 \leq i \neq j \leq q_d)$, where $A_1, \ldots, A_{q_d}$ are the variables used to construct d.

Algorithm to build $CDBS(Q)$

Using the above definitions, we can now show an algorithm that builds the canonical databases of Q.

Algorithm to build $CDBS(Q)$
Input: $Q = q(\boldsymbol{X}) : -p_1(\boldsymbol{X}_1), ..., p_n(\boldsymbol{X}_n), K_1(\boldsymbol{Y}_1), ..., K_m(\boldsymbol{Y}_m)$,
where the K_i's are the built-in predicates.
Output: $\mathcal{CDBS}(Q)$
Method:
Step 1: Let $db(Q) = \{p_1(\boldsymbol{X}_1), ..., p_n(\boldsymbol{X}_n)\}$;
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 $Mappings = \emptyset$;
//
Step 2: // Definition of Q-mappings (using `C` syntax).
for$(i_1 = 1; i_1 \leq q; i_1 + +)$
for$(i_2 = 1; i_2 \leq q; i_2{+}{+})$
$\vdots$
for$(i_q = 1; i_q \leq q; i_q{+}{+})\{$
$\theta_j = (A_{i_1}, A_{i_2}, ..., A_{i_q})$;
if (there is no Q-mapping in $Mappings$ that is isomorphic to θ_j)$\{$
$Mappings = Mappings \cup \{\theta_j\}$;
$j{+}{+}$;
$\}$
$\}$
Step 3: // Now using the Q-mappings we generate the $j - 1$ databases $d_i \in CDBS(Q)$

//
for($l = 1; l < j; l$++)
if($\theta_l(K_1(\boldsymbol{Y}_1)) \wedge \cdots \wedge \theta_l(K_m(\boldsymbol{Y}_m))$ is unsatisfiable $CDB_l = \emptyset$;
else $CDB_l = \theta_l(db(Q))$;
Step 4: return $\{CDB_1, ..., CDB_{j-1}\}$;
Each CDB_i has attached to it the constraints
$(\theta_i(K_1(\boldsymbol{Y}_1)) \wedge \cdots \wedge \theta_i(K_m(\boldsymbol{Y}_m))) \wedge (A_j \neq A_k, \forall j, k\ 1 \leq j \neq k \leq q)$

3.3 Applying Q_2 to a canonical database

Our method to test whether $Q_1 \subseteq Q_2$ first builds $CDBS(Q_1)$, the set of canonical databases, d_i, for Q_1, and then it applies Q_2 to these databases. Let us describe how to apply a query (Q_2) to a canonical database d_i.

First, we have to define what it means that a tuple, defined on the variables used to construct canonical databases, belongs to $Q_2(d)$, where d is a canonical database in $CDBS(Q_1)$.

Definition 1. *Let d be a database in $CDBS(Q_1)$. We say that $t \in Q_2(d)$ if for all ground substitutions α, defined on the variables in d, $\alpha(t) \in Q_2(\alpha(d))$.*

The following lemma gives a condition to test membership of a tuple in $Q_2(d)$.

Lemma 1. *Let Q_1 and Q_2 of the form [11]*
$Q_1 : I$:- $J_1 \wedge J_2 \wedge \cdots \wedge J_l \wedge K_1 \wedge K_2 \wedge \cdots \wedge K_n$.
$Q_2 : H$:- $G_1 \wedge G_2 \wedge \cdots \wedge G_r \wedge F_1 \wedge F_2 \wedge \cdots \wedge F_m$.
where J_i's and G_i's are ordinary subgoals, and K_i's and F_i's are the built-in predicates.

Let $d \in CDBS(Q_1)$. Then $t \in Q_2(d)$ if and only if

1. *There are assignment mappings $\tau_1, \ldots, \tau_k$ from the ordinary subgoals of Q_2 to d such that $\tau_1(H) = \cdots = \tau_k(H) = t$, and*
2. *The formula $F = constraints(d) \wedge \neg(\tau_1(F_1) \wedge \cdots \wedge \tau_1(F_m)) \wedge \cdots \wedge \neg(\tau_k(F_1) \wedge \cdots \wedge \tau_k(F_m))$ is not satisfiable.*

Proof. only-if part: Condition 1 must be true, otherwise t cannot be in $Q_2(d)$. Now by contradiction we prove that condition 2 is also necessary. Consider all such assignment mappings from Q_2 to d. Suppose that F is satisfiable. Then there is a ground substitution α for the variables of F, such that for all j, $1 \leq j \leq k$, $\alpha(\tau_j(F_1) \wedge \cdots \wedge \tau_j(F_m))$ is $false$. Using the constants in the substitution, we can define a database D such that $\alpha(t) \notin Q_2(D)$. Thus, $t \notin Q_2(d)$.
if-part: Assume conditions 1 and 2 hold. Therefore, F is not satisfiable. Since F is not satisfiable, for any substitution α that grounds d, there is a $j, 1 \leq j \leq k$ such that $\alpha(\tau_j(F_1) \wedge \cdots \wedge \tau_j(F_m))$ is *true*; that is, the built-in predicates in Q_2 always hold under this condition. Then for any such α, $\alpha(t) \in Q_2(\alpha(d))$, and hence this part of the lemma follows. □

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

Example 6. Let Q_1 and Q_2 be as defined in Example 1.

One of the canonical databases that we build from Q_1 is $d = \{q(A, B), r(C, D), r(D, C)\}$, where A, B, C, D are uninterpreted constants that satisfy $constraints(d) = (A \neq B) \wedge (A \neq C) \wedge (A \neq D) \wedge (B \neq C) \wedge (B \neq D) \wedge (C \neq D) \wedge (A \geq B)$; this represents a situation where X, Y, U, and V are mapped to different values in the database; another canonical database, $d' = \{q(A, A), r(A, A)\}$, will capture the case when the variables of Q_1 are mapped to the same constant; for this query there are 15 different canonical databases.

For this canonical database d, Q_2 can obtain the tuple $p(A, B)$. There are two ways to map the ordinary subgoals of Q_2 to d to obtain $p(A, B)$: $h_1(X) = h_2(X) = A$, $h_1(Y) = h_2(Y) = B$, $h_1(U) = h_2(V) = C$, $h_1(V) = h_2(U) = D$. Then we use both mappings to test whether $p(A, B)$ belongs to $Q_2(d)$, checking if the following formula is not satisfiable:

$$constraints(d) \wedge \neg(h_1(U \leq V)) \wedge \neg(h_2(U \leq V))$$

If it is not satisfiable, and since we assume that d satisfies $constraints(d)$, it means that at least one of the two constraints $(h_1(U \leq V))$, $(h_2(U \leq V))$ is true.

The above formula is not satisfiable, since

$$constraints(d) \wedge \neg(h_1(U \leq V)) \wedge \neg(h_2(U \leq V))$$

$$=$$

$$constraints(d) \wedge \neg(h_1(U) \leq h_1(V)) \wedge \neg(h_2(U) \leq h_2(V))$$

$$=$$

$$constraints(d) \wedge \neg(C \leq D) \wedge \neg(D \leq C)$$

$$\equiv$$

$$constraints(d) \wedge (C > D) \wedge (D > C)$$

The unsatisfiability of the formula means that either $C \leq D$ or $D \leq C$ is true when we map Q_2 to any grounding of d. Therefore $p(A, B) \in Q_2(d)$. □

3.4 Testing the satisfiability of a formula

The full version of this paper provides the complete and detailed algorithm to test the satisfiability of a formula F of the form:

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

Basically, the algorithm performs the following steps:

1. Normalize the formula, obtaining an equivalent formula in Conjunctive Normal Form without negated atoms.
2. Build a directed graph for each element of the conjunction in the formula. These graphs are similar to the ones produced by the algorithm in [11].

3. Check the satisfiability of each graph. For a graph to be satisfiable, it cannot have cycles of solid arcs and, if the variables range over nondense domains, we must be able to find an ordering of the variables that satisfy the constraints represented by the graph.
4. If none of the graphs is satisfiable, the formula F is unsatisfiable, otherwise it is satisfiable.

4 Main Theorem

The following result is crucial for our test.

Lemma 2. *Let $Q : I :- J_1, \cdots, J_q, K_1, \cdots, K_n$ be a conjunctive query, where the K_i's are its built-in predicates. Let τ be an assignment mapping of Q into a database D, such that $(\tau(K_1) \wedge \cdots \wedge \tau(K_n))$ is true. Let $sd = \{\tau(J_1), \cdots, \tau(J_q)\}$; i.e., sd is the subset of D on which τ maps the ordinary subgoals of Q. Then sd is isomorphic to a database d, where $d \in CDBS(Q)$.*

Proof. The reader can find a complete proof of this lemma in [3, 4]. It takes advantage of the way the canonical databases are built. Given that $CDBS(Q)$ covers all the possible inequalities among variables, the subset sd of D obtained by a mapping of Q will be isomorphic to a canonical database. □

The following theorem shows our condition to test containment of two conjunctive queries with built-in predicates.

Theorem 1. *$Q_1 \subseteq Q_2$ if and only if $\forall d \in CDBS(Q_1)$ ($t_d \in Q_2(d)$).*

Proof. only-if-part: By contradiction. Assume that there is a d, $d \in CDBS(Q_1)$, such that $t_d \notin Q_2(d)$. Then, by definition 1, there exists an 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, $Q_1 \not\subseteq Q_2$,
if-part: Let D be an arbitrary database. Let $t \in Q_1(D)$. We have to prove that $t \in Q_2(D)$.

We have that $t \in Q_1(d)$, so there exists an assignment mapping θ that maps the ordinary predicates of Q_1 to some atoms in D, such that $\theta(I) = t$.

Let $sd = \{\theta(J_1), \cdots, \theta(J_q)\}$, that is, sd is the subset of facts in D mapped by the ordinary subgoals in the body of Q_1 through the assignment mapping θ. By Lemma 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 $t \in Q_2(sd)$. Hence $t \in Q_2(D)$.

So for any database D and for any t in $Q_1(D)$, $Q_2(D)$ contains t. Therefore $Q_1 \subseteq Q_2$. □

4.1 Testing the containment of conjunctive queries with built-in predicates

Our procedure to test the containment of two conjunctive queries with built-in predicates Q_1 and Q_2 consists of the following steps:

1. Build $CDBS(Q_1)$, using the algorithm shown in Section 3.2.
2. Apply Q_2 to all consistent databases d in $CDBS(Q_1)$. Using the different mappings that can reach d from Q_2, and using also $constraints(d)$, build the formula F corresponding to each database.
3. Test if $t_d \in Q_2(d)$ for all d in $CDBS(Q_1)$. To do so, it is only needed to check the unsatisfiability of every formula F. If all formulas are not satisfiable, then $Q_1 \subseteq Q_2$.

Let us illustrate the procedure with an example:

Example 7. Let Q_1 and Q_2 be the following queries:
$Q_1: \ q(X,Y) :- p(X,Y), p(Y,Z), X \geq Y, Y > Z$
$Q_2: \ q(X,Y) :- p(X,Y), p(X,W), X > W$

There is only one mapping τ from Q_2 to Q_1: $\tau(X) = X; \tau(Y) = Y; \tau(W) = Y$. Using the results from Ullman [10, 11], it is not possible to imply $\tau(X > W)$ from the built-in predicates in Q_1, $X \geq Y$ and $Y > Z$. Considering X, Y and Z belonging to a nondense domain like the integers, the results of [8] are not useful either.

However, applying our procedure, which is summarized in Table 6, we can conclude that $Q_1 \subseteq Q_2$. The canonical databases that are possible, consistent with their constraints, are d_2 and d_5. We can apply assignment mappings from Q_2 to each database, and the resulting formulas are not satisfiable. By Lemma 1, we can conclude that $t_{d_2} \in Q_2(d_2)$ and $t_{d_5} \in Q_2(d_5)$. Therefore, we have that for all (consistent) d in $CDBS(Q_1)$, $t_d \in Q_2(d)$. Then, using Theorem 1, we can conclude that $Q_1 \subseteq Q_2$. □

Table 6. Example of our procedure

Q-map.	$CDBS(Q)$		t_d	Mappings	Formula
X Y Z	d_i	p	p		
A A A	d_1	AA			
	$A \geq A \wedge A > A$		d_1 is not consistent		
A A B	d_2	AA	AA	$\tau_1(X) = \tau_1(Y) = A;$	$A \neq B \wedge A > B \wedge$
		AB		$\tau_1(W) = B$	$\neg(A > B) \wedge \neg(A > A)$
	$A \neq B \wedge A \geq A \wedge A > B$			$\tau_2(X) = \tau_2(Y) = A$	*(Not satisfiable)*
				$\tau_2(W) = A$	
A B A	d_3	AB			
		BA	d_3 is not consistent		
	$A \neq B \wedge A \geq B \wedge B > A$				
B A A	d_4	BA			
		AA	d_4 is not consistent		
	$A \neq B \wedge B \geq A \wedge A > A$				
A B C	d_5	AB	AB	$\tau_1(X) = A;$	$A \neq B \wedge B \neq C \wedge$
		BC		$\tau_1(Y) = \tau_1(W) = B$	$A \neq C \wedge A \geq B \wedge$
	$A \neq B \wedge B \neq C \wedge A \neq C$				$B > C \wedge \neg(A > B)$
	$\wedge A \geq B \wedge B > C$				*(Not satisfiable)*

5 Conclusions

We have given an exact characterization to test containment of conjunctive queries with built-in predicates. This solves the long-standing open problem of finding an exact characterization to test this kind of containment over nondense domains.

This result is possible due to our definition and use of Canonical Databases. It is a tool that clearly helps in problems related to containment of queries, as it was already shown in [3, 4] where they had been used to test containment of conjunctive queries under bag-semantics.

References

1. A.V. Aho, Y. Sagiv, and J.D. Ullman. "Equivalence of Relational Expressions". *SIAM J. of Computing 8:2*, pp. 218-246.
2. A.V. Aho, Y. Sagiv, and J.D. Ullman. "Efficient Optimization of a Class of Relational Queries". *ACM TODS 4:4*, pp. 435-454.
3. Brisaboa, N.R., "Inclusión de Consultas Conjuntivas en la semántica de bolsas", Ph.D. Dissertation, Departamento de Computación, Facultade de Informática, Universidade da Coruña, A Coruña, Spain, May 1997.
4. Brisaboa, N.R., Hernández, H.J. Testing bag-containment of conjunctive queries, *Acta Informatica*, 34, 1997. Springer-Verlag, Munich, Germany,pp. 557-578.
5. A.K. Chandra and P.M. Merlin. "Optimal implementation of conjunctive queries in relational databases". *Proc. of 9th ACM Symposium on Theory of Computing*, New York, NY, 1977, pp. 77-90.
6. S. Chaudhuri, M Vardi. "Optimization of Real Conjunctive Queries," *Proc. Twelfth ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems*, Washington, DC, May 1993, pp. 59-70.
7. O. Ibarra, J. Su. "On the containment and equivalence of database queries with linear constraints".*PODS'97*, Tucson, Arizona, 1997,pp. 32–43.
8. Klug, A., "On conjunctive queries containing inequalities," J. ACM **35**:1, 1988, pp. 146-160.
9. J. W. Lloyd. *Foundations of Logic Programming*, Second, Extended edition, Springer-Verlag 1987.
10. Ullman, J. D.: Principles of Database Systems, Second Edition, Computer Science Press, 1982.
11. Ullman, J. D.: Principles of Database and Knowledge-base Systems, Vols. 1-2, Computer Science Press, 1988-1989.