Some algorithmic improvements for the containment problem of conjunctive queries with negation

Michel Leclère and Marie-Laure Mugnier

LIRMM, Université de Montpellier,
161, rue Ada, F-34392 Montpellier cedex - France
{leclere,mugnier}@lirmm.fr

Abstract. Query containment is a fundamental problem of databases. Given two queries q_1 and q_2, it asks whether the set of answers to q_1 is included in the set of answers to q_2 for any database. In this paper, we investigate this problem for conjunctive queries with negated subgoals. We use graph homomorphism as the core notion, which leads us to extend the results presented in [Ull97] and [WL03]. First, we exhibit sufficient (but not necessary) conditions for query containment based on special subgraphs of q_2, which generalize that proposed in [WL03]. As a corollary, we obtain a case where the time complexity of the problem decreases. From a practical viewpoint, these properties can be exploited in algorithms, as shown in the paper. Second, we propose an algorithm based on the exploration of a space of graphs, which improves existing algorithms.

1 Introduction

In this paper, we investigate the problem of deciding on query containment for conjunctive queries with negated subgoals (but without inequalities). *Query containment checking* is one of the fundamental problems of databases. A query q_1 is said to be contained in a query q_2 (notation $q_1 \sqsubseteq q_2$) if for any database instance the set of answers to q_1 is included in the set of answers to q_2. Algorithms based on query containment can be used to solve various problems, such as query evaluation and optimization [CM77] [ASU79], rewriting queries using views [Hal01], detecting independance of queries from database updates [LS93], etc. However, the problem is undecidable for general queries expressed as Datalog programs.

Positive conjunctive queries are a class of frequently used queries which have been investigated since the early days of databases [CM77,Ull89]. Their expressive power is equivalent to the select-join-project queries of relational algebra. Checking containment of positive conjunctive queries is an NP-complete problem. It can be solved by testing the existence of a *query homomorphism* from q_2 to q_1, which maps q_2 to q_1 by substituting its variables by terms (constants or variables) in q_2.

Example 1. Let $q_1 = ans(x, y) \leftarrow r(x, y), r(y, x), p(x, x), s(y)$ and $q_2 = ans(u, v) \leftarrow r(u, v), r(v, w), p(u, w)$ be two conjunctive queries. There is one query homomorphism from q_2 to q_1, which is $h = \{u \rightarrow x, v \rightarrow y, w \rightarrow x\}$. Check that $h(q_2)$ has the same head as q_1 and its body is a part of q_1's body. This proves that $q_1 \sqsubseteq q_2$.

This problem can also be recast as a query evaluation problem by considering the *canonical database* associated with a query. Roughly, this database D^q is obtained from a query q by "freezing" its variables, that is considering them as new elements of the schema domain. Then query containment can be reformulated as evaluating q_2 on D^{q_1} and checking that the set of answers contains the head of q_1 [CM77].

When conjunctive queries are extended to negated subgoals, query containment becomes Π_P^2-complete (Π_P^2 is the class $(co\text{-}NP)^{NP}$). To our best knowledge, only two proposals about algorithms deciding on query containment for this class of queries can be found in the literature. We outline the main points of these proposals here, and will go into further detail later. In [Ull97], Ullman gives the scheme of an algorithm (adapted from a uniform equivalence checking method for Datalog programs [LS93]). This scheme involves generating an exponential number (in the size of q_1) of databases representative of q_1 and evaluating q_2 on these databases. This set of databases can be seen as a generalization of the canonical database of the positive case. A database that does not yield the head of q_1 as an answer to q_2 is a counter-example to the containment.

Example 2. Let $q_1 = ans() \leftarrow r(x,y), s(y,z), p(t), p(z), \neg r(z,t)$ and $q_2 = ans() \leftarrow r(u,v), p(w), \neg r(v,w)$. As $ans()$ has no argument, these queries represent boolean queries. It holds that $q_1 \sqsubseteq q_2$, as will be shown later. In a first step, Ullman's scheme builds the 15 partitions on $\{x,y,z,t\}$, which can be seen as all ways of mapping the variables in q_1 to database values. Each partition yields a database by substituting in q_1 the same value to each set of variables and taking the positive part of the query obtained. For instance, the partition $\{\{x,y\},\{z,t\}\}$ yields the database $\{r(0,0), s(0,1), p(1)\}$. If a database does not make the body of q_1 true, as the database obtained from the partition $\{\{x,z\},\{y,t\}\}$, it is eliminated. In a second step, for each database D, all its extensions obtained by adding tuples using the values and the relation symbols in D, and that still make the body of q_1 true, are considered and it is checked that they yield the substituted head of q_1 as an answer to q_2. For instance, for $D = \{r(0,0), s(0,1), p(1)\}$, all extensions with tuples $r(0,1)$, $r(1,0)$, $s(0,0)$, $s(1,0)$, $s(1,1)$ and $p(0)$ are considered.

In the general case, if v is the number of variables in q_1, a number of databases exponential in v are generated in the first step, then, for each generated database D_i, $2^{((\sum_{r\in R} n_i^{arity(r)})-t)}$ representative databases have to be checked, where R is the set of relation symbols appearing in q_1, n_i is the number of terms in D_i and t is the number of tuples in q_1.

In [WL03], Wei and Lausen exhibit a necessary but not sufficient condition for containment of *safe* queries (which are queries in which all variables appear in positive subgoals): if q_1 is contained in q_2 then there must be a query homomorphism (say h) from the positive part of q_2 (say q_2^+) to the positive part of q_1 (say q_1^+), that does not "contradict" the negative subgoals of q_2 (i.e. for all negative subgoals $\neg p(u)$ in q_2, q_1 does not contain the positive subgoal $p(h(u))$). This property is central to the proposed algorithm. It yields a heuristic for the generation of representative databases, with the aim of concluding sooner from partial representative databases. To check that $q_1 \sqsubseteq q_2$, the algorithm tries to find a query homomorphism (without contradiction) h from q_2^+ to q_1^+, such that for each negative literal $\neg p(u)$ in q_2, either $\neg p(h(u))$ is in q_1 or the query q_1' built from q_1 by adding $p(h(u))$ is such that $q_1' \sqsubseteq q_2$.

Let us outline this algorithm in example 2. There are 2 homomorphisms from $q_2^+ = ans() \leftarrow r(u,v), p(w)$ to q_1^+: $h_1 = \{u \rightarrow x, v \rightarrow y, w \rightarrow z\}$ and $h_2 = \{u \rightarrow x, v \rightarrow y, w \rightarrow t\}$. Both homomorphisms do not contradict any negative subgoal in q_2. Let us consider h_1 and the negative literal $\neg r(v,w)$ in q_2. The idea is that any database answering q_1 that does not contain $r(y,z)$ also answers q_2, thus databases containing $r(y,z)$ have to be checked. $r(y,z)$ is added to q_1, yielding q_1'. There are four query homomorphisms from q_2^+ to ${q'}_1^+$. If it can be concluded that $q_1' \sqsubseteq q_2$, then $q_1 \sqsubseteq q_2$. Otherwise, the homomorphism h_2 has to be considered.

Example 3. (ex. 1.2 in [WL03]) Let $q_1 = ans(x,y) \leftarrow r(x,y), r(y,z), \neg r(x,z)$ and $q_2 = ans(u,w) \leftarrow r(u,v), r(v,w), \neg s(w,w)$. There is one query homomorphism, $h = \{u \rightarrow x, v \rightarrow y, w \rightarrow z\}$, from $q_2^+ = ans(u,w) \leftarrow r(u,v), r(v,w)$ to q_1^+. h does not contradict the negative subgoal of q_2. Then, q_1' is generated from q_1 by adding $s(z,z)$. Again h is the sole homomorphism from q_2^+ to ${q'}_1^+$ but it contradicts $\neg s(w,w)$. Thus, $q_1' \not\sqsubseteq q_2$ and as there is no other homomorphism from q_2^+ to q_1^+ it is concluded that $q_1 \not\sqsubseteq q_2$.

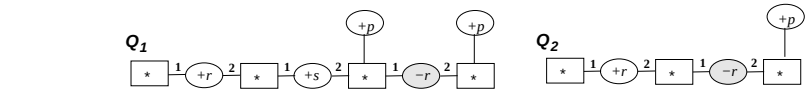
Contribution. In this paper, we consider *homomorphism* as a core notion, where a homomorphism is not restricted to the positive parts of queries as in previous proposals, but extended to whole queries. For this, we propose to view the queries as labeled graphs, called *polarized graphs*. More specifically, a query is represented as a bipartite graph, with two kinds of nodes: relation nodes and terms nodes[1]. Each term of the query becomes a term node, labeled by $\star$ if it is a variable (it can be seen as a "blank node") otherwise by the constant itself. A positive (resp. negative) literal with relation symbol r becomes a relation node labeled by $+r$ (resp. $-r$) and it is linked to the nodes assigned to its terms. The numbers on edges correspond to the position of each term in the literal. See Figure 1, which displays the queries in example 2.

Basically, a homomorphism from an algebraic structure to another maps the elements of the first structure to elements of the second structure while preserving the relations between elements. A homomorphism h from a graph G_2 to a graph G_1 is a mapping from the nodes of G_2 to the nodes of G_1, which preserves edges, that is if xy is an edge of G_2 then $h(x)h(y)$ is an edge of G_1. Since our graphs are labeled, there are additional conditions on labels: a relation node is mapped to a node with the same label; a term node can be mapped to any term node if it is labeled by a $\star$, otherwise it is mapped to a node with the same constant. Numbers on edges are preserved.

Graph homomorphism yields another perspective on queries, as it naturally considers positive and negative occurrences of relations in the same way; moreover, it is defined on subgraphs that do not necessarily correspond to a query, which is convenient for our study. However, let us point out that all definitions and results in this paper can be expressed using the classical vocabulary on queries. In what follows, the term homomorphism can be understood as "query homomorphism extended to negative subgoals" or "graph homomorphism".

A first property, extending the central property in [WL03], is that the existence of a homomorphism from q_2 to q_1 is a sufficient condition for containment.

[1] Queries have often been considered as hypergraphs. The graphs we consider can be seen as the incidence bipartite graphs of these hypergraphs.



$q_1 = ans() \leftarrow r(x, y), s(y, z), p(t), p(z), \neg r(z, t)$
$q_2 = ans() \leftarrow r(u, v), p(w), \neg r(v, w)$

Fig. 1. Queries as graphs

Example 4. Let $q_1 = ans(y, z) \leftarrow r(x, z), r(y, z), \neg r(x, y)$ and $q_2 = ans(u, v) \leftarrow r(u, v), r(w, e)\neg r(w, u)$. There is a homomorphism, say h, from q_2 to q_1, thus $q_1 \sqsubseteq q_2$. $h = \{w \rightarrow x, u \rightarrow y, v \rightarrow z, e \rightarrow z\}$.

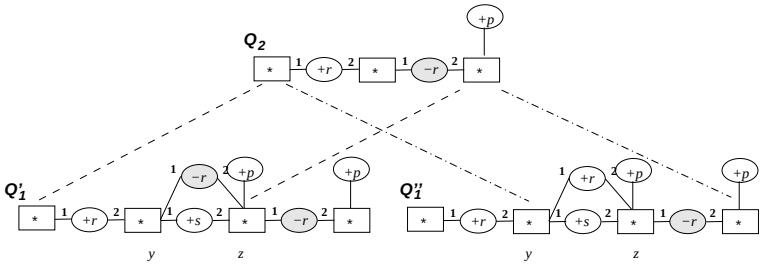


Fig. 2. Graph homomorphisms from Q_2 to Q'_1 and Q''_1

The existence of a homomorphism is not a necessary condition, as can be seen in example 2 (pictured in Figure 1): $q_1 \sqsubseteq q_2$ but there is no homomorphism from q_2 to q_1. However, q_1 can be equivalently rewritten as the union of two queries: one obtained by adding $\neg r(y, z)$, the other by adding $r(y, z)$. These queries are shown in Figure 2. As there is a homomorphism from q_2 to both queries, we conclude that $q_1 \sqsubseteq q_2$.

More generally, instead of considering representative databases, we rewrite q_1 into more and more specific queries. We are lead to consider a space of graphs (or queries) partially ordered by inclusion, with greatest element q_1 and least elements the "complete" graphs, obtained from q_1 by adding as many literals as possible. A brute-force algorithm generates all complete graphs and check that there is a homomorphism from q_2 to each of them.

Roughly, Ullman's scheme can be seen as generating all complete graphs from q_1. We should point out, however, that the first step in computing all partitions on q_1 terms is not necessary, *i.e.* the discrete partition is sufficient. The set of representative databases generated from the discrete partition is the set of complete graphs. Although it is not claimed in [Ull97], Ullman's algorithm is in fact able to process queries with inequalities (see section 2.4).

This framework being settled, we focus on two points. First, we search for cases where the problem is simpler. We study special subgraphs of q_2 that necessarily map to q_1 (theorem 2 and 3); q_2^+ is a specific case. As a corollary, when the whole q_2 satisfies

one of these conditions, the containment problem becomes equivalent to homomorphism checking, thus its time complexity falls into NP (property 8). From a practical viewpoint, these properties can be exploited in algorithms, including Wei and Lausen's algorithm. They can be used in a preprocessing phase (to limit the number of representative databases or to conclude before entering the generation phase) or to guide the construction of representative databases. Second, we propose an algorithm based on exploration of the graph space. This algorithm is simple to describe and implement. Its correctness is directly given by the exhibited properties (theorem 4) and its complexity is analyzed (property 10). We compare this algorithm to Wei and Lausen's algorithm, which can be seen as exploring the same space of graphs but in a radically different way. In particular, our algorithm requires a space polynomial in the size of the initial queries (provided that the maximal arity of a relation is bounded by a constant), which is not the case for Wei and Lausen's algorithm.

The paper is organized as follows. The next section introduces the graph framework and reformulates the query containment problem in terms of graph homomorphism. It ends with a brute-force algorithm, whose complexity is compared to that of Ullman's scheme. Section 3 is devoted to necessary or sufficient conditions for containment. Section 4 presents our algorithm based on space exploration and compares it to Wei and Lausen's algorithm.

2 Preliminary considerations

We first recall basic definitions and results on databases. Then we introduce the graph framework, which leads us to recast CQC as a conjunction of homomorphism tests. We end with a brute-force algorithm, that we compare with Ullman's scheme.

2.1 Basic database notions

A *database schema* $S = (R, dom)$ includes a finite set of relations R and a countably infinite set of constants dom. Each relation has an *arity* (not equal to zero) defining the number of its arguments. A *database instance* D (or simply a *database*) over S maps each *k*-ary relation r_i of R to a finite subset of dom^k (denoted $D(r_i)$). A *conjunctive query (with negation)* is of form:

$$q = ans(u) \leftarrow r_1(u_1),\ ...\ r_n(u_n), \neg s_1(v_1),\ ...\ \neg s_m(v_m)\ \ n \geq 0,\ m \geq 0,\ n + m \geq 1$$

where $r_1\ ...\ r_n, s_1\ ...\ s_m$ are relations, ans is a special relation not belonging to R, u and $u_1\ ...\ u_n, v_1\ ...\ v_m$ are tuples of terms (variables or constants of dom), and each variable of u occurs at least once in the body of the rule. Without loss of generality, we assume that the same literal does not appear twice in the body of the rule. A *positive query* is a query without negative literals ($m = 0$, thus $n \geq 1$). A query is *safe* if each variable occurring in a negative literal also occurs in a positive one. A query is *inconsistent* if it contains two opposite literals (i.e. $\exists\, i, j\ 1 \leq i \leq n,\ 1 \leq j \leq m$ such that $r_i(u_i) = s_j(v_j)$), otherwise it is consistent.

Given a query $q = ans(u) \leftarrow r_1(u_1),\ ...\ r_n(u_n),\ \neg s_1(v_1),\ ...\ \neg s_m(v_m)$ and a database D on S, $q(D)$ denotes the *set of answers* to q in D; $q(D)$ is the set of tuples

$\mu(u)$ where μ is a substitution of the variables in q by constants in dom such that for any i in $\{1, ..., n\}$, $\mu(u_i) \in D(r_i)$ and for any j in $\{1, ..., m\}$, $\mu(v_j) \notin D(s_j)$. We also call μ a substitution *from q to D*. When the arity of ans is 0, $q(D)$ is the set $\{()\}$ if there is such a substitution μ, otherwise it is $\emptyset$. If $q(D)$ is not empty, D is said to *answer* the query.

A query q_1 is said to be *contained* in a query q_2, noted $q_1 \sqsubseteq q_2$, if for any database D, $q_1(D) \subseteq q_2(D)$. The *conjunctive query containment problem* (CQC) takes as input two conjunctive queries q_1 and q_2 and asks whether $q_1 \sqsubseteq q_2$. When q_1 and q_2 are positive, it can be reformulated as a query homomorphism problem, where a homomorphism is defined as follows: a *query homomorphism* from $q = ans(u) \leftarrow r_1(u_1),\ ...\ r_n(u_n)$ to $q' = ans(u') \leftarrow r'_1(u'_1),\ ...\ r'_{n'}(u'_{n'})$ is a substitution θ of the variables in q by terms (variables or constants) in q' such that $\theta(u) = u'$ (thus u and u' have the same size) and for any i in $\{1, ..., n\}$, there is j in $\{1, ..., n'\}$ such that $\theta(r_i(u_i)) = r'_j(u'_j)$. The query homomorphism theorem proves that, given two positive queries q_1 and q_2, $q_1 \sqsubseteq q_2$ iff there is a query homomorphism from q_2 to q_1.

2.2 CQC and homomorphism

As explained in the introduction, it is convenient to see a query as a bipartite labeled graph, that we call a polarized graph (PG)[2]. The mappings between graph and database notions used in this paper are immediate. To represent heads of queries, we use special relations ans_i for each possible arity i, possibly 0 (which corresponds to boolean queries). Then the head of a query (say $ans(t_1...t_k)$) is mapped to a positive relation node with label ans_k and with i-th neighbor the node assigned to t_i. We usually omit ans_0 in drawings (f.i. Figure 1: there is an implicit isolated relation node labeled ans_0 in each graph). It is easily checked that a graph homomorphism from a graph representing a query to another is equivalent to a query homomorphism extended to negative subgoals from the first query to the second (the above definition of a query homomorphism can be used without change if we consider that r_i and r'_j represent possibly negated relation). That is why we use the same term homomorphism for both notions. If there is a homomorphism from q_2 to q_1, we say that q_2 can be *mapped* to q_1. We will keep the term literal and its notation $p(u)$ or $\neg p(u)$, where u is a sequence of terms, to denote a subgraph induced by a relation node and its neighbors.

For positive conjunctive queries q_1 and q_2, $q_1 \sqsubseteq q_2$ iff there is a homomorphism from q_2 to q_1. For conjunctive queries with negation, one part of this property still holds:

Property 1. Given conjunctive queries q_1 and q_2, if there is a homomorphism from q_2 to q_1 then $q_1 \sqsubseteq q_2$.

For the other direction, we assume that q_1 and q_2 are consistent. This assumption will be made in the sequel of the paper. Even if q_1 and q_2 are consistent, we might have $q_1 \sqsubseteq q_2$ and no homomorphism from q_2 to q_1, as illustrated by Figures 1 and 2.

[2] For space limitation reasons, we do not provide here precise definitions concerning PGs. These graphs are a simplification of graphs used in a knowledge representation context, see [Ker01,ML06].

Definition 1. *A consistent query (or a PG) q is* complete *w.r.t. a set of relations R, if for each relation r in R with arity k, for each k-tuple of terms u in q, not necessarily distinct, q contains $r(u)$ or $\neg r(u)$.*

A complete query is obtained from a query q by repeatedly adding positive and negative literals (on terms already present in q), as long as it does not yield a redundancy or an inconsistency. CQC can be expressed as the conjunction of homomorphism checking problems: one for each complete query generated from q_1.

Property 2. Given two conjunctive queries q_1 and q_2 (with q_1 being consistent), $q_1 \sqsubseteq q_2$ iff for each complete query q_1^c generated from q_1, there is a homomorphism from q_2 to q_1^c.

Note that q_2 can be considered as a *connected* graph: indeed, if q_2 is not connected, a homomorphism from q_2 to q_1 is given by a set of homomorphisms from each connected component of q_2 to q_1, and reciprocally.

2.3 A brute force algorithm for CQC

Property 2 yields a brute force algorithm (cf. algorithm 1) for CQC.

Algorithm 1: Brute force CQC Check

```
Data: consistent queries q1 and q2
Result: true if q1 ⊑ q2, false otherwise
begin
    Let B be the set of complete queries obtained from q1 w.r.t. R;
    forall q1^c ∈ B do
        if there is no homomorphism from q2 to q1^c then
            // q1^c is a counter-example
            return false;
    return true;
end
```

Property 3. The time complexity of Algorithm 1 is in $\mathcal{O}(2^{(n_1)^k \times |\mathcal{R}|} \times hom(q_2, q_1^c))$, where n_1 is the number of terms in q_1, k is the maximum arity of a relation, $\mathcal{R}$ is the set of considered relations and $hom(q_2, q_1^c)$ is the complexity of checking the existence of a homomorphism from q_2 to q_1^c.
Its space complexity is in $\mathcal{O}(max(size(q_2), size(q_1), (n_1)^k \times |\mathcal{R}|))$.

Homomorphism checking is NP-complete (but polynomial as soon as q_2 has a tree-like structure). A brute force algorithm solving this problem for q_2 and q_1^c has a time complexity in $\mathcal{O}(min(n_1^{v_2}, r_1^{r_2}))$, where n_1 is the number of term nodes in q_1^c, v_2 is the number of variable nodes in q_2, r_1 and r_2 are the number of literals in q_1^c and q_2 resp. For comparison with other algorithms, it should be noted that the space complexity of

Algorithm 1 is not exponential in the size of q_1 or q_2 but only in the maximum arity of a relation in $\mathcal{R}$. Indeed, as completions can be generated one by one, the space complexity corresponds to the size of one q_1^c.

2.4 Relationships with Ullman's scheme

Ullman's scheme involves the two following steps:

1. Consider all partitions of the variables in q_1. Build a canonical database from each partition as follows: first assign a distinct constant to each set of the partition, then substitute in q_1 each variable by the constant assigned to its set; let q_1' be the substituted query; the canonical database is composed of the positive literals of q_1' body. We obtain D_1 ... D_k canonical databases if k is the number of partitions. Eliminate the D_i which do not make the body of q_1 true, (i.e. the body of q_1' is inconsistent).
2. For each remaining D_i, test whether for each database $D'i$ obtained from D_i by adding tuples on the same symbol set as D_i, and without contradicting negative subgoals of q_1, it holds that $q_2(D_i')$ includes the head of q_1'. If all D_i satisfy the test, then $q_1 \sqsubseteq q_2$, otherwise not.

This scheme can be reformulated as follows in our framework:

1. *Build all consistent queries D_i obtainable from q_1 by merging some variables.*
2. *The test is satisfied iff q_2 can be mapped to all complete queries obtainable from these D_i.*

From property 2, it is clear that step 1 is useless. Indeed, there is a homomorphism from q_1 to each D_i, $1 \leq i \leq k$, q_1 being identical to the D_i obtained with the discrete partition, say D_1. For a given D_i' there is a D_1' with a homomorphism from D_1' to D_i' induced by the partition on the variables in q_1 yielding D_i. It is thus sufficient to test all D_1', *i.e. all complete queries obtainable from* q_1. This observation leads to an important reduction in the number of tested databases/queries: if v is the number of variables in q_1, step 1 builds a number of databases D_i exponential in v (from which only consistent ones are kept) and each remaining D_i leads in turn to an exponential test (see Algorithm 1).

Step 1 would be necessary if the queries would contain inequalities as in [LS93]. However in [Ull97] and further papers dealing with queries without inequalities, it seems that the uselessness of step 1 had not be noticed.

3 Necessary / sufficient conditions for containment

This section studies conditions that are necessary or sufficient for containment. These properties can be used as filtering properties leading to conclude before entering the generation phase. They can also be used to reduce the number of graphs generated either because they eliminate relations that are not needed in the completion or because they guide the incremental generation of complete graphs (see the next section). Besides their practical algorithmic interest, they also yield particular cases where the theoretical complexity of CQC decreases.

3.1 Immediate properties on labels

Let us begin by considering the node labels. An immediate observation is that if a constant or a relation label (that is a relation with a given polarity) in q_2 does not appear in q_1, then $q_1 \not\sqsubseteq q_2$. A second observation is that relations that do not appear in both q_1 and q_2 are not needed in the completion of q_1. The next property takes the polarity of their occurrences into account.

Property 4. If r is a relation that does not have both positive and negative occurrences in q_2, then r is not needed in the completion of q_1 (i.e. $q_1 \sqsubseteq q_2$ iff q_2 can be mapped to each completion of q_1 built without considering r).

Proof. ($\Leftarrow$) If q_2 can be mapped to each complete query without considering r then $q_1 \sqsubseteq q_2$. Indeed, let q_1^c be any complete query built from q_1. Let $q_1^{c-\{r\}}$ be obtained from q_1^c by removing all literals, occurrences of r, that do not belong to q_1. There is a natural homomorphism, say h_1 from $q_1^{c-\{r\}}$ to q_1^c. By hypothesis there is a homomorphism, say h, from q_2 to $q_1^{c-\{r\}}$. The composition of these homomorphisms $h_1 \circ h$ is a homomorphism from q_2 to q_1^c.

($\Rightarrow$) Let $q_1 \sqsubseteq q_2$. Assume that $q_1^{c-\{r\}}$ is a completion (without considering r) of q_1 such that there is no homomorphism from q_2 to $q_1^{c-\{r\}}$. We show that this assumption leads to contradict $q_1 \sqsubseteq q_2$. If all the occurrences of r in q_2 are positive (resp. negative), let q_1^{c-} (resp. q_1^{c+}) be the complete query obtained from $q_1^{c-\{r\}}$ by adding solely negative (resp. positive) literals with relation r. Since $q_1 \sqsubseteq q_2$ there is a homomorphism from q_2 to q_1^{c-} (resp. q_1^{c+}). This homomorphism necessarily maps all occurrences of r in q_2 into q_1; more generally, no literal of q_2 can be mapped to the added negative (resp. positive) occurrences of r. h is thus a homomorphism from q_2 to $q_1^{c-\{r\}}$, which contradicts the hypothesis. □

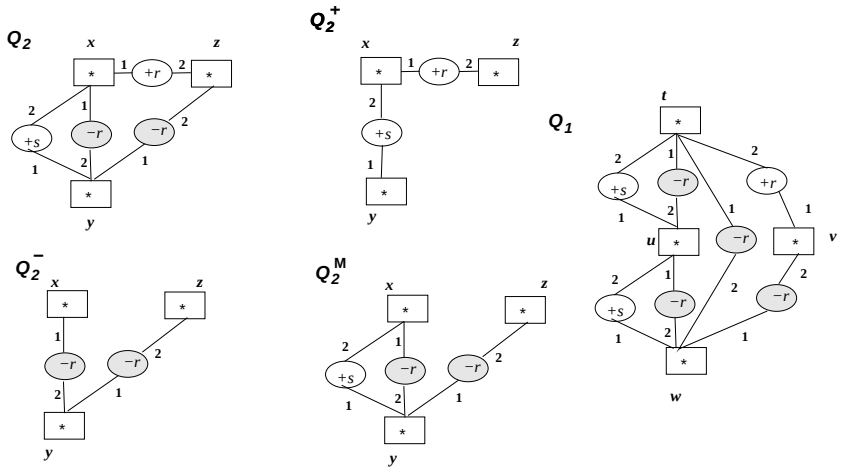
As a corollary to the previous properties, we obtain:

Theorem 1. *$q_1 \sqsubseteq q_2$ iff q_2 can be mapped to each completion of q_1 w.r.t. relations occurring in both positive and negative forms in q_1 and q_2.*

Let us consider the queries in example 3: as $\neg s$ does not appear in q_1, it can be immediately concluded that $q_1 \not\sqsubseteq q_2$. Would $\neg s(w, w)$ not exist in q_2, as r does not appear positively and negatively both in q_1 and q_2, no relation can be used for completion, thus there is also immediate failure.

3.2 Special subgraphs

As we have seen, a homomorphism from q_2 to q_1 is a sufficient but not necessary condition for containment. The objective here is to identify parts - or subgraphs - of q_2 (i.e. conjunctions of literals appearing in q_2) for which there must be a homomorphism to q_1. Moreover, such a homomorphism from a subgraph of q_2 to q_1 has to be potentially extensible to a homomorphism from the entire q_2 to a completion of q_1. We call it a *compatible* homomorphism. See Figure 3: there are three homomorphisms from q_2^- to q_1: $h_1 = \{x \to t, y \to u, z \to w\}$, $h_2 = \{x \to t, y \to w, z \to v\}$,



$q_2 = ans() \leftarrow s(y,x) \wedge r(x,z) \wedge \neg r(x,y) \wedge \neg r(y,z)$
$q_1 = ans() \leftarrow s(u,t) \wedge r(t,v) \wedge s(w,u) \wedge \neg r(u,w) \wedge \neg r(t,u) \wedge \neg r(w,v) \wedge \neg r(t,w)$

Fig. 3. Pure subgraphs of q_2 ($q_1 \sqsubseteq q_2$)

$h_3 = \{x \to u, y \to w, z \to v\}$. To check the compatibility, we have to consider $s(y,x)$ and $r(x,z)$. h_1 is not compatible because it leads to map $r(x,z)$ to $\neg r(t,w)$.

Definition 2 (compatible homomorphism). *Given two queries q_2 and q_1, and q'_2 any subgraph of q_2 defining a well-formed PG (i.e. q'_2 is any conjunction of literals appearing in q_2), a homomorphism h from q'_2 to q_1 is said to be* compatible *(w.r.t. q_2) if for each literal of q_2 that does not appear in q'_2 but has all its terms in q'_2, say $t_1...t_k$, there is no literal with the same relation and the opposite polarity on $h(t_1)...h(t_k)$ in q_1.*

Given a query q, the *positive subgraph* of q, denoted by q^+ is the subgraph obtained from q by selecting only the positive literals of q. The *negative subgraph* q^- of q is the dual notion, that is the subgraph obtained from q by selecting only the negative literals of q. See q_2^+ and q_2^- in Figure 3. The next property is the same as theorem 1 in [WL03] reformulated and proven in the graph framework, except that we extend the definition of q^+ to non-safe queries. Note that, when q is a safe query, q^+ contains all term nodes of q.

Property 5. [WL03] If there is no compatible homomorphism from q_2^+ to q_1 (or equivalently to q_1^+), then $q_1 \not\sqsubseteq q_2$.

Proof. Let q_1^{c-} be the negative completion of q_1. If $q_1 \sqsubseteq q_2$ then there is a homomorphism h from q_2 to q_1^{c-}, which necessarily maps q_2^+ to q_1^+. Let $l_i = \neg r(t_1...t_k)$ be any negative literal of q_2. Since h is a homomorphism, q_1^{c-} contains a literal $\neg r(h(t_1)...h(t_k))$. As q_1 is consistent, it cannot contain a literal $r(h(t_1)...h(t_k))$. We conclude that h with domain restricted to q_2^+ is a compatible homomorphism to q_1. □

A similar property is obtained by considering q_2^- instead of q_2^+.

Property 6. If there is no compatible homomorphism from q_2^- to q_1 (or equivalently to q_1^-), then $q_1 \not\sqsubseteq q_2$.

Proof. Consider q_1^{c+} the positive completion of q_1 instead of q_1^{c-}. □

Both q_2^+ and q_2^- notions can be generalized in the following way.

Definition 3 (q_2^{max} pure subgraph). *A pure subgraph of q_2 is a subgraph that does not contain opposite occurrences of the same relation. We note q_2^{max} a pure subgraph of q_2 that is maximal for inclusion.*

Observe that a q_2^{max} is obtained from q_2 by selecting, for each relation appearing in q_2, either all its positive occurrences or all its negative occurrences. See Figure 3: q_2 has two pure subgraphs maximal for inclusion; q_2^+ and q_2^M. q_2^+ (resp. q_2^-) is the particular case where positive (resp. negative) occurrences are chosen for all relations; but it is not necessarily maximal for inclusion as a relation may appear only negatively (res. positively). The ans_i relation is a particular case of such relation.

Theorem 2. *If there is a q_2^{max} that cannot be mapped by a compatible homomorphism to q_1, then $q_1 \not\sqsubseteq q_2$.*

Proof. Consider q_1^{-max} as the completion of q_1 built as follows: for each relation r, if r occurs positively (resp. negatively) in q_2^{max} then complete q_1 with negative (resp. positive) occurrences of r. If q_2^{max} cannot be mapped to q_1 by a compatible homomorphism, then it cannot be mapped by a compatible homomorphism to q_1^{-max} (since by construction no literal of q_2^{max} can be mapped to an added literal). Since q_2 cannot be mapped to q_1^{-max}, q_1^{-max} is a counter-example to the property $q_1 \sqsubseteq q_2$. □

This theorem can be slightly extended by taking into account the occurrences of terms in the literals.

Definition 4. *Two literals are said to be* dependant *if (1) they have an opposite polarity, (2) they have the same relation and (3) their atoms are unifiable after a renaming of their common variables. Otherwise they are said to be* independant.

Two atoms are not unifiable after a renaming of their common variables if their unification would lead to unify different constants. For instance, let a and b be distinct constants; $r(x, a)$ and $\neg r(y, b)$ are independant literals; $p(x, x, a)$ and $\neg p(b, y, y)$ are independant literals as well, whereas $r(x, a)$ and $\neg r(b, y)$ are dependant literals.

Definition 5. *An independant subgraph of a query q_2 is a subgraph of q_2 composed of pairwise independant literals.*

More generally, let us say that two literals are "exchangeable" if they can have the same list of images for their arguments by homomorphisms to (necessarily distinct) completions of q_1. F.i. given the distinct constants a and b, the literals $r(x, a)$ and $\neg r(b, y)$ are dependant but, if q_1 contains $r(a, b)$, they are not exchangeable. Independant subgraphs, and a fortiori pure subgraphs, are only particular cases of subgraphs without exchangeable literals; the general notion of "exchangeability" remains to be studied. Exchangeable literals are responsible for the problem complexity, as shown by the next property.

Property 7. If $q_1 \sqsubseteq q_2$, then there is a compatible homomorphism from every subgraph of q_2 composed of pairwise non-exchangeable literals to q_1.

Sketch of proof. Consider such a subgraph q' of q_2. Let q_1^{c+} be the completion of q_1 with solely positive literals. If there is no homomorphism from q' to q_1, then for each homomorphism from q' to q_1^{c+}, there is at least one added literal, say $p(u)$, such that a literal $p(v)$ in q' is mapped to $p(u)$. Let us replace *all* such $p(u)$ by $\neg p(u)$. Let $q_1^{c'}$ be the graph obtained. Let h be a homomorphism from q' to $q_1^{c'}$ (there is such a homomorphism since $q_1 \sqsubseteq q_2$). h maps a literal $\neg p(w)$ in q' to a literal $\neg p(u)$, otherwise there would be a homomorphism from q' to q_1. By construction, there is a literal $p(v)$ mapped to $p(u)$ by a homomorphism from q' to q_1^{c+}, thus $p(v)$ and $\neg p(w)$ are exchangeable literals. □

The following extension to the theorem 2 is a corollary of property 7.

Theorem 3. *[Extension to the theorem 2] If there is an independant subgraph of q_2 that cannot be mapped by a compatible homomorphism to q_1, then $q_1 \not\sqsubseteq q_2$.*

We thus obtain a case for which CQC has the same complexity as homomorphism checking:

Property 8. If q_2 is an independant subgraph, then $q_1 \sqsubseteq q_2$ iff there is a homomorphism from q_2 to q_1.

3.3 Filtering implementation

Let us end this section with an implementation of some filtering properties (algorithm 2 that will be used next section).

Algorithm 2: Filtering

Data: consistent queries q_1 and q_2
Result: true, false or undetermined; if true then $q_1 \sqsubseteq q_2$; if false then $q_1 \not\sqsubseteq q_2$
begin
Test 1 **if** *there is a label (r) or ($\neg$r) or a constant occurring in q_2 but not in q_1* **then**
return *false*
Test 2 **if** *there is a homomorphism from q_2 to q_1* **then**
return *true*
Test 3 Let q_2^{Max} be an independant subgraph of q_2 with maximum cardinality;
if *there is no compatible homomorphism from q_2^{Max} to q_1* **then**
return *false*
return *undetermined*;
end

Roughly, Test 1 is in $\mathcal{O}(rlog_2(r))$ where r is the maximum number of relations in q_1 or q_2. Test 2 and Test 3 perform a homomorphism check. For Test 3, choosing a subgraph with maximum size is an obvious choice but there may be other criteria f.i. based on the structure of the obtained subgraph. Alternatively, one can choose to check several or all independant subgraphs instead of one.

4 Space exploration algorithm

The space of queries "leading" from q_1 to its completions is structured in a sup-semi-lattice by graph inclusion (given two queries q_1 and q_2 in this space, $q_1 \leq q_2$ if q_2 is a subgraph of q_1). The question "is there a homomorphism from q_2 to each q_1^c (completion of q_1)" can be reformulated as follows "is there a *covering set*, that is a subset of incomparable queries of this space $\{q_1, \ ..., \ q_k\}$ such that (1) there is a homomorphism from q_2 to each q_i ; (2) for each q_1^c there is a q_i with $q_1^c \leq q_i$.

The brute-force algorithm (Algorithm 1) takes the set of all completions of q_1 as covering set. The next algorithm (Algorithm 3 and recursive search Algorithm 4) searches the space in a top-down way starting from q_1 and tries to build a covering set with partial completions of q_1. Case-based reasoning is applied at each step: for a given relation r with arity k and a tuple $(t_1...t_k)$ such that neither $r(t_1...t_k)$ nor $\neg r(t_1...t_k)$ is present in the current partial completion, two queries are generated according to each case. The algorithm is justified by the following property:

Theorem 4. *$q_1 \sqsubseteq q_2$ if and only if:*
1. There is a homomorphism h from q_2 to q_1 **or**
2. $q' \sqsubseteq q_2$ and $q'' \sqsubseteq q_2$ where q' (resp. q'') is obtained from q_1 by adding the positive literal $r(t_1...t_k)$ (resp. the negative literal $\neg r(t_1...t_k)$) where r is a relation of arity k occurring in q_2 (both in positive and negative forms) and $t_1...t_k$ are terms of q_1 such that neither the literal $r(t_1...t_k)$ nor the literal $\neg r(t_1...t_k)$ is already present in q_1.

Proof (sketch). ($\Rightarrow$) By recurrence on the number of literals to add to q_1 to obtain a complete query. ($\Leftarrow$) Condition 1 corresponds to property 1. For condition 2, see that $\{q', q''\}$ is a covering set. □

Subalgorithm 4 is supposed to have direct access to data available in the main algorithm 3. The choice of r and $t_1...t_k$, in Algorithm 4, can be guided by a compatible homomorphism from an independant graph.

Algorithm 3: Check by space exploration

```
Data: Consistent queries q1 and q2
Result: true if q1 ⊑ q2, false otherwise
begin
    Result ← Filtering(); // See Algorithm 2
    if (Result ≠ undetermined) then
        return Result
    Let R^{+−} be the set of relation names occurring in both negative and positive forms in
    q2;
    return RecCheck(q1); // See Algorithm 4
end
```

The following property ensures that Algorithm 4 does not generate the same query several times, which is a crucial point for complexity. Otherwise the algorithm could be worse than the brute-force algorithm in the worse-case.

Algorithm 4: RecCheck(q)

Data: Consistent query q **Access:** q_2, $\mathcal{R}^{+-}$
Result: true if $q \sqsubseteq q_2$, false otherwise
begin
  if *there is a homomorphism from* q_2 *to* q **then**
    return *true*
  if q *is complete w.r.t.* $\mathcal{R}^{+-}$ **then**
    return *false*
  `/* Test 3 of filtering can be reused in each call */`
  $(r, t_1...t_k) \leftarrow$ **ChooseLiteralsToAdd**(q);
  `/* r is a relation of` $\mathcal{R}^{+-}$ `and` $t_1...t_k$ `are terms of q */`
  Let q' be obtained from q by adding the literal $r(t_1...t_k)$;
  Let q'' be obtained from q by adding the literal $\neg r(t_1...t_k)$;
  return *(***RecCheck***(*q'*)* AND **RecCheck***(*q''*))*
end

Property 9. The subspace explored by Algorithm 4 is a (binary) tree.

Indeed, at each recursive call, $\{q',\ q''\}$ is a covering set inducing a bipartition of the covered space: each query in this space is below exactly one of these two queries.

Property 10. Algorithm 3 has the same time and space complexity as Algorithm 1.

Proof. Property 9 ensures that the number of queries generated is at most twice the number of completions of q_1 (in the worse case, the complete queries are the leaves of the generated tree of queries). Checking whether a query is complete can be done in constant time if the number of literals in the query is incrementally maintained. Thus time complexity is the same as Algorithm 1. For space complexity, see that the tree is explored in a depth-first way. □

Wei and Lausen's algorithm is based on the following theorem (theorem 2 of their paper reformulated in graph terms; moreover, in (1) "compatible" has been added, as well as step (2.1) to prevent inconsistent queries to be built[3]). This theorem considers *safe* queries (otherwise h could be undefined on some variables in q_2).

Theorem 5. *[WL03] With q_1 and q_2 being safe queries, $q_1 \sqsubseteq q_2$ if and only if:*
1. There is a compatible homomorphism h from q_2^+ to q_1^+, such that:
2. for each negative literal $l_i = \neg r(t_1...t_k)$ in q_2, (2.1) either h can be extended to include l_i **or** *(2.2) $q'_i \sqsubseteq q_2$ holds, where q'_i is obtained from q_1 by adding the positive literal $r(h(t_1)\ ...\ h(t_k))$.*

Note that if each negative literal l_i can be included in h then h is a homomorphism from q_2 in q_1. An important point is that this theorem induces a radically different way of searching the space than that of Algorithm 3. Indeed, whereas Algorithm 3 develops a tree, condition (2) leads to build a covering set that does not partition the space. An algorithm applying this property directly is thus likely to explore the same subspaces several times.

[3] Indeed, the theorem does not apply to inconsistent queries. If q_1 is inconsistent, it is by definition included in any q_2, but there might be no homomorphism from q_2^+ to q_1^+.

The algorithm proposed by Wei and Lausen (in the appendix of their paper) sketchily proceeds as follows. First, all homomorphisms from q_2^+ to q_1^+ are generated. Then for each compatible homomorphism, say h, and for each negative literal that cannot be mapped by extending h, a new query to test is generated from q_1 by adding a positive literal according to the previous theorem. This algorithm can be seen as developing a *and/or* tree: a homomorphism h leads to success if all queries q'_i directly generated from it lead to containment; a query q'_i leads to containment if there is a homomorphism from $q_i'^+$ leading to success. The and/or tree is traversed in a breadth-first manner.

This algorithm has a space complexity exponential in the size of the initial queries, at least because all homomorphisms from q_2^+ to q_1^+ are first generated and the and/or tree is traversed in a breadth-first manner. Concerning time complexity, the key question is whether the same query can be generated several times. The notion of "new" mapping is mentioned in the algorithm (when the homomorphisms from q_2^+ to q_1^+ are enumerated, only *new* mappings are retained) but without detail about how a "new" mapping is recognized. *A priori* one has to store all already generated mappings to recognize a new one. If so, the space complexity would be exponential in the size of q_2 even with the assumption that homomorphisms are generated one by one and the tree is traversed in a depth-first way. To summarize, the algorithm we propose in this paper (see Algorithms 3 and 4) has the following qualities compared to Wei and Lausen's algorithm:

- it is not restricted to safe queries;
- the space exploration is based on special subgraphs, which generalize the q_2^+ notion (and could be used instead of it in condition 1 of Wei and Lausen's theorem);
- it is polynomial in space (if the arity of relations is bound by a constant);
- it is simple to describe and implement.

Acknowledgments. We specially thank a referee for his/her valuable comments.

References

[ASU79] A. V. Aho, Y. Sagiv, and J. D. Ullman. Equivalences among relational expressions. *SIAM J. Comput.*, 8(2):218–246, 1979.

[CM77] A.K. Chandra and P.M. Merlin. Optimal implementation of conjunctive queries in relational databases. In *9th ACM Symposium on Theory of Computing*, pages 77–90, 1977.

[Hal01] A. Y. Halevy. Answering queries using views: A survey. *VLDB Journal: Very Large Data Bases*, 10(4):270–294, 2001.

[Ker01] G. Kerdiles. *Saying it with Pictures: a Logical Landscape of Conceptual Graphs*. PhD thesis, Univ. Montpellier II / Amsterdam, Nov. 2001.

[LS93] A. Y. Levy and Y. Sagiv. Queries independent of updates. In *VLDB*, pages 171–181, 1993.

[ML06] M.L. Mugnier and M. Leclère. On querying simple conceptual graphs with negation. *Data and Knowledge Engineering (DKE)*, 2006. In press, doi:10.1016/j.datak.2006.03.008.

[Ull89] J. D. Ullman. *Principles of Database and Knowledge-Base Systems, Volume II*. Computer Science Press, 1989.

[Ull97] J. D. Ullman. Information Integration using Logical Views. In *International Conference on Database Theory (ICDT)*, 1997.

[WL03] F. Wei and G. Lausen. Containment of Conjunctive Queries with Safe Negation. In *International Conference on Database Theory (ICDT)*, 2003.