Computing Complete Answers to Queries in the Presence of Limited Access Patterns (Revision)*

Chen Li
Department of Information and Computer Science
University of California at Irvine, CA 92697-3425
chenli@ics.uci.edu
Tel: 1-949-824-9470, Fax: 1-949-824-4056

Abstract

In data applications such as information integration, there can be limited access patterns to relations, i.e., binding patterns require values to be specified for certain attributes in order to retrieve data from a relation. As a consequence, we cannot retrieve all tuples from these relations. In this article we study the problem of computing the *complete* answer to a query, i.e., the answer that could be computed if all the tuples could be retrieved. A query is *stable* if for any instance of the relations in the query, its complete answer can be computed using the access patterns permitted by the relations. We study the problem of testing stability of various classes of queries, including conjunctive queries, unions of conjunctive queries, and conjunctive queries with arithmetic comparisons. We give algorithms and complexity results for these classes of queries. We show that stability of datalog programs is undecidable, and give a sufficient condition for stability of datalog queries. Finally, we study data-dependent computability of the complete answer to a nonstable query, and propose a decision tree for guiding the process to compute the complete answer.

Keywords: limited access patterns to relations, complete answers to queries, query stability.

1 Introduction

Traditional database systems answer user queries using data stored in relations, assuming all the data can be retrieved. In several recent database applications, relations do not support complete scans of their data. Instead, there are limited access patterns to these relations, i.e., they have binding patterns that require values to be specified for certain attributes in order to retrieve data from a relation. Thus we cannot retrieve all their tuples. For example, the goal of data integration [C+94, HKWY97, IFF+99, LRO96, MAM+98, YÖL97] is to support seamless access to heterogeneous data sources. In heterogeneous environments, especially in the context of World Wide Web, sources have diverse and limited query capabilities. For instance, many Web movie sources such as The IMDB [IMD] and Cinemachine [Cin] provide search forms for movie information. A user fills out a form

*This article combines and integrates the technical report [Li99] and some content in the paper presented in the 8th International Conference on Database Theory (ICDT), London, UK, January, 2001 [LC01b]. In addition to the prior materials, this article contains more results and complete proofs that were not included in the original reports.

by specifying the values of some attributes, e.g., movie title, or star name, and the source returns information about movies satisfying the query conditions.

In this article we study the following fundamental problems: given a query on relations with binding restrictions, can its complete answer be computed? If so, what is the execution plan? The *complete* answer to a query is the answer that could be computed if we could retrieve all the tuples from the relations in the query. Computing the complete answer to a user query is important for decision support and analysis by the user. However, due to the relation restrictions, we can retrieve only part of the relations, and we must do some reasoning about the completeness of the answer computed by a plan. The following example shows that, nevertheless, in some cases the complete answer to a query can be computed.

EXAMPLE 1.1 Suppose we have two relations $r(Star, Movie)$ and $s(Movie, Award)$ that store information about movies and their stars, and information about movies and the awards they won, respectively. The access limitation of relation r is that each query to this relation must specify a star name. Similarly, the access limitation of s is that each query to s must specify a movie name. Consider the following query that asks for the awards of the movies in which Fonda starred:

```
SELECT Award
FROM   r, s
WHERE  Star = 'fonda' AND r.Movie = s.Movie;
```

This query can be written as the following conjunctive query [Ull89]

$$Q_1 : ans(A) \text{ :- } r(fonda, M), s(M, A)$$

To answer Q_1, we first access relation r to retrieve the movies in which Fonda starred. For each returned movie, we access relation s to obtain its awards. Finally we return all these awards as the answer to the query. Although we did not retrieve all the tuples in the two relations, we can still claim that the computed answer *is* the complete answer to Q_1. The reason is that all the tuples of relation r that satisfy the first subgoal were retrieved in the first step. In addition, all the tuples of s that satisfy the second subgoal and join with the results of the first step were retrieved in the second step.

However, if the access limitation of relation r is that each query to r must specify a movie title (not a star name), then we cannot compute the complete answer to query Q_1. The reason is that there can be a movie that Fonda starred, but we cannot retrieve the tuple without knowing the movie. □

In general, if the complete answer to a query can be computed for any database of the relations in the query, we say that the query is *stable*. Query Q_1 is stable. As illustrated by the example, we might think that we can test the stability of a conjunctive query by checking the existence of a *feasible* order of all its subgoals, as in [FLMS99, YLUGM99]. An order of subgoals is feasible if for each subgoal in the order, the variables bound by the previous subgoals provide enough bound arguments that the relation for the subgoal can be accessed using a legal pattern. However, the following example shows that a query can be stable even if such a feasible order does *not* exist.

EXAMPLE 1.2 We modify query Q_1 slightly by adding a subgoal $r(S, M)$, and have the following query:

$$Q_2 : ans(A) \text{ :- } r(fonda, M), s(M, A), r(S, M)$$

This query, which seems unnatural, can be generated by a view-expansion process at mediators such as TSIMMIS [LYV+98]. Q_2 does not have a feasible order of all its subgoals, since we cannot bind the variable S in the added subgoal. However, this subgoal is actually redundant, and we can show that Q_2 is equivalent to query Q_1. That is, they produce the same answer for any database. Therefore, for a database, we can compute the complete answer to Q_2 by answering Q_1. This example suggests that testing stability of a conjunctive query is not just checking the existence of a feasible order of all its subgoals. □

These two examples show a subtle difference between the feasibility and the stability of a query. The feasibility depends on the "form" the query is written, while the stability might be related to whether the query has an equivalent that is feasible. The feasibility does not necessarily imply the stability. In this article we study how to test stability of a variety of queries. We define the problem formally in Section 2. The following are the results:

1. In Section 3 we show that a conjunctive query is stable if and only if its minimal equivalent query has a feasible order of all its subgoals. We propose two algorithms for testing stability of conjunctive queries, and prove this problem is $\mathcal{NP}$-complete.

2. In Section 4 we presents results on stability of finite unions of conjunctive queries, and give similar results as conjunctive queries. We propose two algorithms for testing stability of unions of conjunctive queries.

3. In Section 5 we give an algorithm for testing stability of conjunctive queries with arithmetic comparisons.

4. In Section 6 we show that stability of datalog programs is undecidable, and give a sufficient condition for stability of datalog queries.

5. For a nonstable query, in some cases we may still compute its complete answer, even though the computability is data dependent. In Section 7 we discuss in what cases the complete answer to a nonstable conjunctive query may be computed. We develop a decision tree that guides the planning process to compute the complete answer to a conjunctive query.

1.1 Related Work

Several works have considered binding restrictions in the context of answering queries using views [DL97, LMSS95, Qia96]. Rajaraman, Sagiv, and Ullman [RSU95] proposed algorithms for answering queries using views with binding patterns. In that paper all solutions to a query compute the complete answer to the query; thus only stable queries are handled. Duschka and Levy [DL97] solved the same problem by translating source restrictions into recursive rules in a datalog program to obtain the maximally-contained rewriting of a query, but the rewriting does not necessarily compute the query's complete answer. Notice in [RSU95, DL97], binding patterns can be on views that are defined over base relations, while in our paper binding patterns are over the base relations themselves.

A query on relations with binding restrictions can be generated by a view-expansion process at mediators as in TSIMMIS. [LYV+98] studied the problem of generating an executable plan based on source restrictions. [FLMS99, YLUGM99] studied query optimization in the presence of binding restrictions. [YLGMU99] considered how to compute mediator restrictions given source restrictions.

These studies did not consider the possibility that removing subgoals may make an infeasible query feasible. Thus, they regard the query Q_2 in Example 1.2 as an unsolvable query, thus miss the chances of computing its complete answer. [LC00, LC01a] studied how to compute the maximal answer to a conjunctive query with binding restrictions by borrowing bindings from relations not in the query. The paper focused on how to trim irrelevant relations that do not help in obtaining bindings. However, the computed answer may not be the complete answer. As we will see in Section 7, we can sometimes use the approach in these papers to compute the complete answer to a nonstable conjunctive query. Other works on computing answers to queries given incomplete source data include [FKL97, GGH98, KL88, Lev96, MM01], and these studies do not consider limited access patterns to relations.

The dynamic case of computing a complete answer to a nonstable query, as illustrated in Section 7, is different from the case of dynamic mediators discussed in [YLGMU99]. In [YLGMU99], source descriptions can specify a set of values that can be bound to an attribute at a source. The uncertainty of whether the mediator can answer a query comes from the fact that, before the query is executed, it is unknown whether the intermediate bindings are allowed by a source. In this article we use bound-free adornments to describe relation restrictions. As we will see in Section 7, without executing a plan, we do not know whether the tuples for the nonanswerable subgoals can join with all the tuples in the supplementary relations [BR87] of the answerable subgoals. Thus the computability of the complete answer is data dependent.

2 Problem Formulation

In this section, we formalize the problem studied in this article. We use *binding patterns* of relations to model their limited access patterns [Ull89]. A binding pattern of a relation specifies the attributes that must be given values ("bound") to access the relation. In each binding pattern, an attribute is adorned as "*b*" (a value must be specified for this attribute) or "*f*" (the attribute can be free). For instance, the limitation of relation $r(Star, Movie)$ in Example 1.1 is represented as a binding pattern *bf*, meaning that each query to the relation must provide a $Star$ name. The limitation of the relation $s(Movie, Award)$ is also represented as binding pattern *bf*. In general, a relation can have more than one binding pattern. For example, a relation $p(A, B, C)$ with two binding patterns *bff* and *ffb* requires that every query to the relation must either supply a value for the first attribute A, or supply a value for the third attribute C.

Given a database D of relations with binding patterns and a query Q on these relations, the *complete* answer to Q, denoted by $ANS(Q, D)$, is the query's answer that could be computed if we could retrieve *all* tuples from the relations. However, we may not be able to retrieve all these tuples due to the binding patterns. The following observation serves as a starting point of our work.

> If a relation does not have an all-free binding pattern, then after some finite source queries are sent to the relation, there can always be some tuples in the relation that have not been retrieved, because we did not obtain the necessary bindings to retrieve them.

Definition 2.1 (**stable query**) A query on relations with binding patterns is *stable* if for any database of the relations, we can compute the complete answer to the query by accessing the relations with legal patterns, i.e., by providing constants that meet the requirement of at least one binding pattern. □

We assume that if a relation requires a value to be given for a particular attribute, the domain of the attribute is infinite, or it is impossible to enumerate all possible values for this attribute. For instance, we assume there are infinite number of A values satisfying a condition $1 \leq A \leq 3$, i.e., values of attribute A are not just integers. As another example, the relation $r(Star, Movie)$ in Example 1.1 requires a star name in each query. Under our assumption, we do not know all the possible star names. As a result, we do not allow the strategy of trying all the infinite possible strings as the argument to test the relation. The reasons are (1) this approach does not terminate; and (2) trying arbitrary strings cannot guarantee to retrieve tuples from relations. Instead, we assume that each binding we use to access a relation is either from a given user query, or from the tuples retrieved by another access to a relation, while the value is from the appropriate domain.

Now the fundamental problem we study in this article can be stated as follows: *how to test the stability of a query on relations with binding patterns?* As we will see in Section 3, in order to prove a query is stable, we can just show a legal plan that can compute the complete answer to the query for any database of the relations. On the other hand, to prove that a query Q is not stable, we need to give two databases D_1 and D_2 that have the same observable tuples. That is, by using only the bindings from the query and the relations, for both databases we can retrieve the same tuples from the relations. However, the two databases yield different answers to query Q, i.e., $ANS(Q, D_1) \neq ANS(Q, D_2)$. Therefore, based on the retrievable tuples from the relations, we cannot tell whether the answer computed using these tuples is the complete answer or not.

3 Stability of Conjunctive Queries

In this section we study stability of conjunctive queries. We propose two algorithms for testing stability of conjunctive queries, and show this problem is $\mathcal{NP}$-complete.

A conjunctive query (CQ for short) is denoted by:

$$h(\bar{X}) \text{ :- } g_1(\bar{X}_1), \ldots, g_n(\bar{X}_n)$$

In each subgoal $g_i(\bar{X}_i)$, predicate g_i is a relation, and every argument in $\bar{X}_i$ is either a variable or a constant. The variables $\bar{X}$ in the head are called *distinguished* variables. We use names beginning with lower-case letters for constants and relation names, and names beginning with upper-case letters for variables.

3.1 Feasible Conjunctive Queries

Definition 3.1 (**feasible order of subgoals**) Some subgoals $g_{i_1}(\bar{X}_{i_1}), \ldots, g_{i_k}(\bar{X}_{i_k})$ in a CQ form a *feasible order* if each subgoal $g_{i_j}(\bar{X}_{i_j})$ in the order is *executable*; that is, there is a binding pattern p of the relation g_{i_j}, such that for each argument A in $g_{i_j}(\bar{X}_{i_j})$ that is adorned as b in p, either A is a constant, or A appears in a previous subgoal. A CQ is *feasible* if it has a feasible order of *all* its subgoals. □

The query Q_1 in Example 1.1 is feasible, since $[r(fonda, M), s(M, A)]$ is a feasible order of all its subgoals. The query Q_2 is not feasible, since it does not have such a feasible order. A subgoal in a CQ is *answerable* if it is in a feasible order of *some* subgoals in the query. The answerable subgoals of a CQ can be computed by a greedy algorithm, called "Inflationary" algorithm, which works as follows.

Initialize a set Φ_a of subgoals to be empty. With the variables bound by the subgoals in Φ_a, whenever a subgoal becomes executable by accessing its relation, add this subgoal to Φ_a. Repeat this process until no more subgoals can be added to Φ_a, and Φ_a will include all the answerable subgoals of the query. Clearly a query is feasible if and only if all its subgoals are answerable. The following lemma shows that feasibility of a CQ is a sufficient condition for its stability.

Lemma 3.1 *A feasible CQ is stable.* □

Proof: Let a feasible CQ Q have a feasible order $g_1(\bar{X}_1), \ldots, g_n(\bar{X}_n)$ of all its subgoals. For any database D, we can compute $ANS(Q, D)$ by executing the following *linear plan*. Compute the corresponding sequence of n *supplementary relations* [BR87, Ull89] $I_1, \ldots, I_n$, where I_i is the supplementary relation after the first i subgoals have been processed. Return the supplementary relation I_n.

Now we prove this linear plan computes $ANS(Q, D)$. For each tuple $t \in ANS(Q, D)$, suppose that t comes from the tuples $t_1, \ldots, t_n$ of the relations $g_1, \ldots, g_n$, respectively. For $j = 1, \ldots, n$, the tuple $t_1 \bowtie \cdots \bowtie t_{j-1}$ is included in I_{j-1} after its values for the irrelevant variables are dropped. (A variable is irrelevant if it does not appear in a later subgoal or the head of the query.) This tuple agrees with the tuple t_j on their common variables. Therefore, during the computation of the supplementary relation I_j in the linear plan, no matter which binding pattern of the relation g_j is chosen, tuple t_j in g_j is retrieved by an access to relation g_j. Based on the way I_j is computed, I_j also includes the tuple $t_1 \bowtie \cdots \bowtie t_j$ after the values for the irrelevant variables are dropped. Thus the supplementary relation I_n computed by the linear plan includes the tuple t, which can be derived from $t_1 \bowtie \cdots \bowtie t_n$ by dropping the values for the nondistinguished variables. ■

Lemma 3.1 shows that the computability of the complete answer to a feasible CQ is *static*, because no matter what the relations mentioned in the query are, the complete answer can be computed by the same linear plan. As we will see in Section 7, the computability of the complete answer to a nonstable CQ is *dynamic*, since the computability is unknown until some plan is executed. Here a plan could be any plan that uses legal patterns of relations, including an exhaustive plan discussed in Section 7.1.

3.2 Minimal Equivalents of Conjunctive Queries

Definition 3.2 (**query containment and equivalence**) A query Q_1 is *contained* in a query Q_2, denoted $Q_1 \sqsubseteq Q_2$, if for any database D, the set of answers to Q_1 is a subset of the answers to Q_2, i.e., $ANS(Q_1, D) \subseteq ANS(Q_2, D)$. The two queries are *equivalent*, denoted $Q_1 \equiv Q_2$, if $Q_1 \sqsubseteq Q_2$ and $Q_2 \sqsubseteq Q_1$. □

Chandra and Merlin [CM77] showed that for two CQs Q_1 and Q_2, $Q_1 \sqsubseteq Q_2$ if and only if there is a *containment mapping* from Q_2 to Q_1. Example 1.2 in Section 1 shows that even if a query is not feasible, it can still be stable, since its minimal equivalent may be feasible. A CQ is *minimal* if it has no redundant subgoals, i.e., removing any subgoal from the query will yield a nonequivalent query. It is known that each CQ has a unique minimal equivalent up to renaming of variables and reordering of subgoals, which can be obtained by deleting its redundant subgoals [CM77]. Thus Lemma 3.1 can be strengthened to the following corollaries.

Corollary 3.1 *A conjunctive query is stable if it has an equivalent query that is feasible.* □

Corollary 3.2 *If a CQ has a minimal equivalent that is feasible, then the query is stable.* □

However, if the minimal equivalent Q_m of a CQ Q is not feasible, it is still not clear whether there exists an equivalent query that is feasible. In analogy with [RSU95], we might need to consider the possibility that by adding some redundant subgoals to query Q_m, we could have an equivalent query that is feasible. In principle, we have to consider all the equivalents of the query Q to check whether some of them are feasible. Note that there are infinite number of equivalents to a query, and some of them may look quite different from the query. Fortunately, we have the following lemma.

Lemma 3.2 *If a minimal CQ is not feasible, then it has no equivalent that is feasible.* □

Proof: Let Q be a minimal CQ that is not feasible. Suppose there is an equivalent CQ P that is feasible, and $\Theta_P = \langle e_1, \ldots, e_m \rangle$ is a feasible order of all the subgoals in P. Since the two queries are equivalent, there exist two containment mappings μ: $Q \rightarrow P$, and ν: $P \rightarrow Q$. Consider the targets in Q of the subgoals $e_1, \ldots, e_m$ under the mapping ν: $\nu(e_1), \ldots, \nu(e_m)$. Scan these subgoals from $\nu(e_1)$ to $\nu(e_m)$, and remove the subgoals with identical subgoals earlier in the sequence, and we have an order of *some* subgoals in Q, say, $\Theta_Q = \langle g_1, \ldots, g_n \rangle$, as shown in Figure 1. Now we prove that Θ_Q is a feasible order of *all* the subgoals in Q. That is, we need to show: (1) Θ_Q includes all the subgoals in query Q; (2) Θ_Q is a feasible order. Since query Q is not feasible, we can claim that the equivalent query P actually does not exist.

$$\begin{array}{llcccc}
Q\colon & H \text{ :- } g_1(\ldots), \ldots, & g_t(\ldots, X, \ldots), & \ldots, & g_i(\ldots, X, \ldots), & \ldots, g_n(\ldots) \\
\mu \downarrow \; \uparrow \nu & & \uparrow \nu & & \uparrow \nu & \\
P\colon & E \text{ :- } e_1(\ldots), \ldots, & e_j(\ldots, Y, \ldots), & \ldots, & e_{k_i}(\ldots, Y, \ldots), & \ldots, e_m(\ldots)
\end{array}$$

Figure 1: Proof of Lemma 3.2

Claim (1) is correct because Q is minimal. If Θ_Q did not include all the subgoals in Q, let query Q' have the head of Q and the subgoals in Θ_Q. Then $Q' \sqsubseteq P$ because of the containment mapping ν, and $P \sqsubseteq Q'$ because of the containment mapping μ. Thus Q' is equivalent to Q, and Q could not be minimal.

We now prove claim (2). Consider the first subgoal $g_1 = \nu(e_1)$ in Θ_Q. Since the containment mapping ν maps a variable to a variable or a constant, and maps a constant to the same constant, all the targets of the constant arguments in subgoal e_1 must also be constant arguments in subgoal g_1. Since e_1 is answerable by the relation e_1, subgoal g_1 is also answerable by the relation g_1, which is the same as relation e_1.

Now consider each subgoal g_i in the order Θ_Q, and let $g_i = \nu(e_{k_i})$ for some $1 \leq k_i \leq m$. Since subgoal e_{k_i} is answerable in Θ_P, there is a binding pattern p of relation e_{k_i}, such that for each argument Y in subgoal e_{k_i} that is adorned b in binding pattern p, either Y is a constant, or Y is a variable bound by a previous subgoal e_j. Consider the argument $X = \nu(Y)$ in subgoal g_i. If Y is a constant, then X is also a constant. If Y is a variable, and X is not a constant, based on how Θ_Q was constructed, there exists a subgoal g_t before g_i in Θ_Q, such that $g_t = \nu(e_j)$. Therefore, the variable X is also bound by the subgoal g_t. In summary, X is either a constant or a variable that is bound by a previous subgoal in Θ_Q. Subgoal g_i satisfies the binding requirements of the binding pattern p, and thus it is also answerable by the relation g_i. ∎

Lemma 3.3 *If the minimal equivalent of a CQ is not feasible, then the query is not stable.* □

Proof: Assume that the minimal equivalent Q_m of a query Q is not feasible. In order to prove that Q is not stable, we construct two databases D_1 and D_2, such that $ANS(Q, D_1) \neq ANS(Q, D_2)$, but D_1 and D_2 have the same observable tuples. Since we cannot tell which database we have by looking at the observable tuples, no plan for the query can guarantee that its computed answer is the complete answer to the query.

Let $X_1, \ldots, X_m$ be all the variables in Q_m. Each variable X_i is assigned a *distinct* value x_i. All the relations are empty initially. For each subgoal g_i in Q_m, add a tuple t_i to its relation as follows. For each argument A in subgoal g_i, if A is a constant c, then the corresponding component in t_i is c. If A is a variable X_j, then the corresponding component in t_i is the distinct value x_j assigned to this variable. Let $\Phi_a = \{g_1, \ldots, g_k\}$ be the set of answerable subgoals of Q_m, and $\Phi_{na} = \{g_{k+1}, \ldots, g_n\}$ be the set of nonanswerable subgoals. (See Section 3.1 for the definition of answerable subgoals.) Since Q_m is not feasible, Φ_{na} is not empty. Let this substitution turn the head of Q_m to a tuple t_h.

$$
\begin{array}{lll}
Q_m: & H \text{ :- } \overbrace{g_1, \ldots, g_k}^{\Phi_a}, \overbrace{g_{k+1}, \ldots, g_n}^{\Phi_{na}} \\
D_1: & \quad t_1, \ldots, t_k \\
D_2: & \quad t_1, \ldots, t_k, \; t_{k+1}, \ldots, t_n
\end{array}
$$

Figure 2: Proof of Lemma 3.3

As shown in Figure 2, D_1 is constructed by adding the tuples $t_1, \ldots, t_k$ to the relations $g_1, \ldots, g_k$, respectively; D_2 is constructed by adding all the tuples $t_1, \ldots, t_n$ to the relations $g_1, \ldots, g_n$, respectively.[1] A relation may have multiple tuples, since it may appear in multiple subgoals of Q_m. Under both databases we can retrieve tuples $t_1, \ldots, t_k$ following a feasible order of the subgoals $g_1, \ldots, g_k$. Under database D_2, we cannot obtain the necessary bindings to retrieve the tuples $t_{k+1}, \ldots, t_n$. Thus D_1 and D_2 have the same observable tuples, i.e., the tuples in D_1. Clearly $t_h \in ANS(Q_m, D_2)$. Now we only need to prove that $t_h \notin ANS(Q_m, D_1)$.[2] Otherwise, there must be a substitution τ from a subset of the obtainable tuples $\{t_1, \ldots, t_k\}$ to all the subgoals $g_1, \ldots, g_n$, such that under τ each subgoal becomes true. Let Q'_m be a query with the head of Q_m plus the subgoals of the tuples used in τ. Since each variable was assigned with a distinct constant, these constants can represent their corresponding variables. Thus τ can be considered to be a containment mapping from Q_m to Q'_m, and $Q_m \equiv Q'_m$. Then Q_m could not be minimal, since it has an equivalent query Q'_m that has fewer subgoals. ∎

In general, if we want to prove a query Q is not stable, we need to show two databases D_1 and D_2, such that $ANS(Q, D_1) \neq ANS(Q, D_2)$, but these two databases have the same observable tuples.

3.3 Two Algorithms for Testing Stability of Conjunctive Queries

By Corollary 3.2 and Lemma 3.3 we have the following theorem.

[1]Tuples $t_1, \ldots, t_n$ are called *canonical tuples* of query Q.

[2]Note that this claim might not be correct if Q_m is not minimal.

Theorem 3.1 *A conjunctive query is stable if and only if its minimal equivalent is feasible.* □

This theorem suggests an algorithm *CQstable* for testing the stability of a CQ, as shown in Figure 3.

Algorithm CQstable: Test stability of CQs.
Input: • Q: A conjunctive query.
• B: Binding patterns of the relations used in Q.
Output: Decision about the stability of Q.
Method:
(1) Compute the minimal equivalent Q_m of Q by deleting its redundant subgoals.
(2) Based on B, use the algorithm Inflationary to test the feasibility of the query Q_m.
(3) If Q_m is feasible, then Q is stable; otherwise, Q is not stable.

Figure 3: Algorithm: CQstable

Let us analyze the complexity of this algorithm. Assume that a CQ Q has n subgoals, and its minimal equivalent Q_m has k subgoals. It is known that the minimization of CQs is $\mathcal{NP}$-complete [CM77], so the first step takes exponential-time in the size of query Q. A number of papers (e.g., [ASU79a, ASU79b, JK83, Sar91]) considered special cases that have polynomial-time algorithms to minimize queries. The complexity of the algorithm Inflationary in the second step is $O(k^2)$. Since $k \leq n$, the total time complexity of the algorithm CQstable is exponential in the size of Q.

The exponential complexity of the algorithm CQstable comes from the fact that we need to minimize a CQ before testing its feasibility. There is a more efficient algorithm that is based on the following results.

Definition 3.3 (**answerable subquery**) For a CQ Q on relations with binding restrictions, its *answerable subquery*, denoted Q_a, is the CQ that has Q's head and answerable subgoals. □

Theorem 3.2 *Let Q be a CQ on relations with binding restrictions. Let Q_a be its answerable subquery. Then Q is stable iff $Q \equiv Q_a$, i.e., they are equivalent as queries.* □

Proof: *If*: straightforward, since query Q_a is a stable query, and it has a feasible order of all its subgoals). *Only if*: The proof is essentially the same as the proof of Lemma 3.3, which is correct as long as the following conditions are satisfied: all subgoals in Q'_m are answerable, and $Q'_m \not\equiv Q_m$. ■

Theorem 3.2 gives another algorithm *CQstable** for testing stability of CQs, as shown in Figure 4. One advantage of this algorithm is that we do not need to minimize a CQ Q if all its subgoals are answerable. Note that if Q is stable, its answerable subquery Q_a may properly include the subgoals in Q's minimal equivalent, since some redundant subgoals in Q might be answerable. In addition, the complexity of step (1) is $O(n^2 \cdot p)$, where n is the number of subgoals in Q, and p is the maximal number of binding patterns for all subgoals. However, if not all the subgoals are answerable in step (1), we still need to check the existence of a containment mapping from Q to Q_a. Another advantage of algorithm CQstable*, as we will see in Section 5, is that we can extend it to CQs with arithmetic comparisons (CQACs for short). We cannot extend the algorithm CQstable to the case of CQACs, since a CQAC does not necessarily have a unique minimal form.

Algorithm CQstable*: Test stability of CQs.
Input: • Q: A conjunctive query.
• B: Binding patterns of the relations used in Q.
Output: Decision about the stability of Q.
Method:
(1) Compute the answerable subquery Q_a: use the algorithm Inflationary to find all the answerable subgoals of Q. Let Q_a be the query with these answerable subgoals and the head of Q.
(2) Check whether these is a containment mapping from Q to Q_a.
(3) If such a containment mapping exists, then Q is stable; otherwise, Q is not stable.

Figure 4: Algorithm: CQstable*

3.4 Complexity of Testing Stability of CQs

We might want to find a polynomial-time algorithm for testing stability of CQs. Unfortunately, the following theorem shows that this problem is $\mathcal{NP}$-complete.

Theorem 3.3 *Testing stability of conjunctive queries is $\mathcal{NP}$-complete.* □

$$\mu \quad \begin{array}{ll} Q: & ans(\bar{X}) \text{ :- } g_1(\bar{X}_1), \ldots, g_k(\bar{X}_k), \ldots, g_n(\bar{X}_n) \\ Q': & ans(\bar{X}) \text{ :- } g_1(\bar{X}_1), \ldots, g_k(\bar{X}_k) \end{array}$$

(a) Testing a containment mapping from Q to Q'

$$\nu \quad \begin{array}{ll} P: & ans(\bar{X}) \text{ :- } h(A), g_1(A, \bar{X}_1), \ldots, g_k(A, \bar{X}_k), g_{k+1}(B, \bar{X}_{k+1}), \ldots, g_n(B, \bar{X}_n) \\ P': & ans(\bar{X}) \text{ :- } h(A), g_1(A, \bar{X}_1), \ldots, g_k(A, \bar{X}_k) \end{array}$$

(b) Testing the stability of P

Figure 5: Proof of Theorem 3.3

Proof: Given a CQ Q and a CQ Q' that is a subset of the subgoals in Q, the problem of deciding whether $Q' \sqsubseteq Q$ is known to be $\mathcal{NP}$-complete [CM77]. We reduce this problem to the problem of testing the stability of a CQ.

Let Q and Q' be the queries in Figure 5 (a), where $k < n$. We construct a query P on relations with binding restrictions, such that P is stable iff $Q' \sqsubseteq Q$. Figure 5 (b) shows how P is constructed. Let A and B be two new variables that do not appear in the subgoals of Q. For each relation g_i, introduce a new relation g'_i with one more attribute than g_i, and g'_i has only one binding pattern *bff* $\ldots$ *f*. Introduce a new monadic (i.e., 1-ary) relation h with a binding pattern *f*. Let P be the query with the same head of Q and subgoals $h(A), g'_1(A, \bar{X}_1), \ldots, g'_k(A, \bar{X}_k)$, $g'_{k+1}(B, \bar{X}_{k+1}), \ldots, g'_n(B, \bar{X}_n)$.

Clearly the above construction of query P takes time that is polynomial in the size of Q. By the construction, the answerable subgoals of P are $h(A), g'_1(A, \bar{X}_1), \ldots, g'_k(A, \bar{X}_k)$. Let P' be the query with these answerable subgoals. By Theorem 3.2, P is stable iff $P \equiv P'$, i.e., there is a containment mapping ν from P to P'. It is easy to show that ν exists iff there is a containment mapping μ from Q to Q', which is true iff $Q' \sqsubseteq Q$. Note that any such containment mapping from P to P' must map both variables A and B to A. ∎

4 Stability of Unions of Conjunctive Queries

In this section we study stability of unions of CQs, and present results similar to those for CQs. In particular, we show that a union of CQs is stable iff each query in its minimal equivalent is stable. We also propose two algorithms for testing stability of unions of CQs.

Let $\mathcal{Q} = Q_1 \cup \cdots \cup Q_n$ be a finite union of conjunctive queries (UCQ for short), all of which have a common head predicate. It is known that there is a unique minimal subset of $\mathcal{Q}$ that is its minimal equivalent [SY80]. The following theorem is from [SY80].

Theorem 4.1 *Let $\mathcal{Q} = Q_1 \cup \cdots \cup Q_m$ and $\mathcal{R} = R_1 \cup \cdots \cup R_n$ be two UCQs. Then $\mathcal{Q} \sqsubseteq \mathcal{R}$ (i.e., $\mathcal{Q}$ is contained in $\mathcal{R}$ as queries) iff for any query Q_i in $\mathcal{Q}$, there is a query R_j in $\mathcal{R}$, such that $Q_i \sqsubseteq R_j$.* □

EXAMPLE 4.1 Let us see some examples of UCQs and their stability. Suppose that we have three relations r, s, and p, and each relation has only one binding pattern *bf*. Consider the following three queries.

$$
\begin{array}{lll}
Q_1: & ans(X) & \text{:- } r(a, X) \\
Q_2: & ans(X) & \text{:- } r(a, X), p(Y, Z) \\
Q_3: & ans(X) & \text{:- } r(a, X), s(X, Y), p(Y, Z)
\end{array}
$$

Clearly $Q_3 \sqsubseteq Q_2 \sqsubseteq Q_1$. Queries Q_1 and Q_3 are both stable (since they are both feasible), while query Q_2 is not. Consider the following two UCQs: $\mathcal{Q}_1 = Q_1 \cup Q_2 \cup Q_3$ and $\mathcal{Q}_2 = Q_2 \cup Q_3$. $\mathcal{Q}_1$ has a minimal equivalent Q_1, and $\mathcal{Q}_2$ has a minimal equivalent Q_2. Therefore, query $\mathcal{Q}_1$ is stable, and $\mathcal{Q}_2$ is not. □

This example suggests that: (1) The fact that a CQ is stable does not imply that its contained queries must also be stable (see queries Q_1 and Q_2); (2) The fact does not imply that the query's containing queries must also be stable (see queries Q_3 and Q_2). One observation from the example is that adding a subgoal to a nonstable query may yield a stable query. In particular, as shown by queries Q_2 and Q_3, the subgoal $s(X, Y)$ can help bind more variables, and link the two "disconnected" subgoals $r(a, X)$ and $p(Y, Z)$. On the other hand, adding a subgoal to a stable query may also yield a nonstable query. For instance, as shown by queries Q_1 and Q_2, the subgoal $p(Y, Z)$ brings a variable Z that cannot be bound by the other subgoal.

4.1 Algorithm: UCQstable

In analogy with Theorem 3.1, we have a theorem that suggests a stability test for UCQs.

Theorem 4.2 *Let $\mathcal{Q}$ be a UCQ on relations with binding restrictions. $\mathcal{Q}$ is stable iff each query in the minimal equivalent of $\mathcal{Q}$ is stable.* □

Proof: Let $\mathcal{Q} = Q_1 \cup \cdots \cup Q_n$. Without loss of generality, assume its minimal equivalent $\mathcal{Q}_m = Q_1 \cup \cdots \cup Q_k$, where $k \leq n$. The "*If*" part is straightforward, since for any database D, we can compute $ANS(\mathcal{Q}, D)$ by computing $ANS(Q_i, D)$ for each Q_i $(1 \leq i \leq k)$, and taking the union of these answers.

Now let us prove the "*Only If*" part. Suppose $\mathcal{Q}_m$ has a query, say Q_1, that is not stable. Without loss of generality, suppose that subgoals $g_1(\bar{X}_1), \ldots, g_p(\bar{X}_p)$ are the answerable subgoals of query Q_1, and subgoals $g_{p+1}(\bar{X}_{p+1}), \ldots, g_q(\bar{X}_q)$ are its nonanswerable subgoals. Let Q_1' be the answerable subquery of Q_1 with the p answerable subgoals. By Theorem 3.2, $p < q$, and $Q_1' \not\equiv Q_1$. Consider the canonical tuples $t_1, \ldots, t_k$ of query Q_1 (see Section 3 for the definition of canonical tuples). Let these tuples turn the head of Q_1 to a tuple t_h.

Following the same idea in the proof of Theorem 3.2, to prove that $\mathcal{Q}$ is not stable, we show that given the obtainable tuples $t_1, \ldots, t_p$, the answer to $\mathcal{Q}$ (i.e., the answer to $\mathcal{Q}_m$) does not include tuple t_h. Suppose that the answer to $\mathcal{Q}_m$ does include tuple t_h. By Theorem 3.2, query Q_1 cannot yield t_h, since otherwise there will be a containment mapping from Q_1' to Q_1, contradicting the fact that $Q_1 \not\equiv Q_1'$. Therefore, tuple t_h must be derived from another query Q_j in $\mathcal{Q}_m$. By the construction of the canonical tuples $t_1, \ldots, t_p$, it is easy to argue that Q_j produces t_h only if there is a symbol mapping from the variables of Q_j to these p tuples. This mapping also serves as a containment mapping from Q_j to Q_1'. Thus, $Q_1' \sqsubseteq Q_j$, and $Q_1 \sqsubseteq Q_1' \sqsubseteq Q_j$. Then $\mathcal{Q}_m$ cannot be minimal, since it has a redundant query Q_1. ∎

Theorem 4.2 gives an algorithm *UCQstable* for testing stability of UCQs, as shown in Figure 6.

Algorithm UCQstable: Test stability of unions of conjunctive queries.
Input: • $\mathcal{Q}$: A finite union of conjunctive queries.
• B: Binding patterns of the relations used in $\mathcal{Q}$.
Output: Decision about the stability of Q.
Method:
(1) Compute the minimal equivalent $\mathcal{Q}_m$ of $\mathcal{Q}$ by deleting its redundant queries.
(2) For each $Q_i \in \mathcal{Q}_m$:
• Use the algorithm CQstable or CQstable* to test the stability of Q_i;
• If Q_i is not stable, then query $\mathcal{Q}$ is not stable.
(3) Claim that query $\mathcal{Q}$ is stable.

Figure 6: Algorithm: UCQstable

4.2 Algorithm: UCQstable*

Similar to Theorem 3.2, we have the following theorem.

Theorem 4.3 *Let $\mathcal{Q}$ be a UCQ on relations with binding restrictions. Let $\mathcal{Q}_s$ be the union of all the stable queries in $\mathcal{Q}$. Then $\mathcal{Q}$ is stable iff $\mathcal{Q} \equiv \mathcal{Q}_s$, i.e., $\mathcal{Q}$ and $\mathcal{Q}_s$ are equivalent as queries.* □

Proof: *If*: Straightforward. If each CQ in $\mathcal{Q}_s$ is stable, for any database D, we can compute $ANS(\mathcal{Q}, D)$ by computing $ANS(Q_i, D)$ for each query $Q_i \in \mathcal{Q}_s$, and taking the union of these answers.

Only If: Suppose $\mathcal{Q}$ is stable, and $\mathcal{Q} \not\equiv \mathcal{Q}_s$. Then there is a nonstable query Q_u in $\mathcal{Q} - \mathcal{Q}_s$, such that Q_u is not contained in any query in $\mathcal{Q}_s$. Since containment between CQs is transitive, there must be a most containing query Q_i in $\mathcal{Q} - \mathcal{Q}_s$, such that Q_i is not contained in any query in $\mathcal{Q}_s$, and there is no query in $\mathcal{Q} - \mathcal{Q}_s$ that properly contains Q_i. Query Q_i can either be Q_u, or can be found by searching all the queries in $\mathcal{Q} - \mathcal{Q}_s$ that contain Q_u, and choosing the most containing one.

Let Q'_i be the answerable subquery of Q_i. Since Q_i is not stable, by Theorem 3.2, $Q_i \not\equiv Q'_i$, and $Q_i \sqsubset Q'_i$. Consider the canonical tuples for the subgoals in Q_i. Assume that these tuples turn the head of Q_i to a tuple t_h. Following the same idea in the proof of Theorem 4.2, to prove that $\mathcal{Q}$ is not stable, we show that given the obtainable tuples for the answerable subgoals in Q_i, we cannot compute the tuple t_h. Otherwise, there can be three cases:

1. Tuple t_h is derived from Q_i, which can not be true by Theorem 3.2.
2. Tuple t_h is derived from a query Q_k in $\mathcal{Q}_s$. Then there is a symbol mapping from the variables of Q_k to these obtainable tuples. This mapping also serves as a containment mapping from Q_k to Q'_i. Therefore, $Q_i \sqsubseteq Q'_i \sqsubseteq Q_k$, contradicting the fact that Q_i is not contained in any CQ in $\mathcal{Q}_s$.
3. Tuple t_h is derived from a query Q_k in $\mathcal{Q} - \mathcal{Q}_s$. Then $Q_i \sqsubset Q'_i \sqsubseteq Q_k$, contradicting to the fact that no query in $\mathcal{Q} - \mathcal{Q}_s$ can properly contain Q_i.

Therefore, $\mathcal{Q}$ is stable only if $\mathcal{Q} \equiv \mathcal{Q}_s$. ∎

Theorem 4.3 gives another algorithm *UCQstable** for testing stability of UCQs, as shown in Figure 7. The advantage of this algorithm is that we might avoid testing the equivalence between the query $\mathcal{Q}_s$ and $\mathcal{Q}$ if all the queries in $\mathcal{Q}$ are stable.

Algorithm UCQstable*: Test stability of unions of conjunctive queries.
Input: • $\mathcal{Q}$: A finite union of conjunctive queries.
• B: Binding patterns of the relations used in $\mathcal{Q}$.
Output: Decision about the stability of Q.
Method:
(1) Compute all the stable queries:
• $\mathcal{Q}_s = \phi$;
• For each query Q_i in $\mathcal{Q}$:
(a) Call the algorithm CQStable or CQStable* to test the stability of Q_i;
(b) If Q_i is stable, add Q_i to $\mathcal{Q}_s$.
(2) Test whether $\mathcal{Q} \sqsubseteq \mathcal{Q}_s$ as queries.
(3) If $\mathcal{Q} \sqsubseteq \mathcal{Q}_s$, then $\mathcal{Q}$ is stable; otherwise, $\mathcal{Q}$ is not stable.

Figure 7: Algorithm: UCQstable*

One corollary of Theorems 4.2 and 4.3 is that stability of bounded datalog queries [CGKV88] is decidable. The reason is that by definition, a bounded datalog program is equivalent to a UCQ. We can test the stability of a bounded datalog query by testing the stability of its equivalent UCQ. It is known that boundedness of datalog programs is undecidable [GMSV93]. Several papers (e.g., [Ioa85, NS87, Sag85]) gave algorithms for detecting boundedness in several classes of datalog queries. Another corollary of the two theorems is that stability of a query with conjuncts and disjuncts is also decidable, since such a query can be translated into an equivalent UCQ. For instance, suppose we have a query with a condition

$$((Author = smith) \vee (Year = 1999)) \wedge (Subject = database).$$

We can rewrite the condition to a disjunctive form:

$$((Author = smith) \wedge (Subject = database)) \vee ((Year = 1999) \wedge (Subject = database)).$$

Therefore, we test the stability of the original query by testing the stability of the corresponding UCQ.

5 Stability of Conjunctive Queries with Arithmetic Comparisons

In this section we study stability of CQs with arithmetic comparisons (CQACs for short), and give algorithms for testing stability.

Let Q be a CQAC. Let $O(Q)$ be the set of ordinary (uninterpreted) subgoals of Q that do not have comparisons. Let $\beta(Q)$ be the set of its subgoals that are arithmetic comparisons. We consider the following arithmetic comparisons: $<$, $\leq$, $>$, $\geq$, and $\neq$. In addition, we make the following assumptions about the comparisons:

1. Values for the variables in the comparisons are chosen from an infinite, totally ordered set, such as the rationals or reals.
2. The comparisons are not contradictory, i.e., there exists an instantiation of the variables such that all the comparisons are true. All the comparisons are safe, i.e., each variable in the comparisons appears in some ordinary subgoal.
3. The comparisons do not imply equalities. The fix is easy. If an equality $X = Y$ is implied by the comparisons, we can rewrite the query by substituting X for Y.

We first review the following fundamental result on containment of CQACs [ALM02].[3]

Theorem 5.1 *Q_1 and Q_2 are two CQACs, and their ACs do not imply "=" restrictions. Let $\mu_1, \ldots, \mu_k$ be all the containment mappings from $O(Q_1)$ to $O(Q_2)$. Then $Q_2 \sqsubseteq Q_1$ if and only if:*

$$\beta(Q_2) \Rightarrow \mu_1(\beta(Q_1)) \vee \ldots \vee \mu_k(\beta(Q_1))$$

i.e., $\beta(Q_2)$ logically implies (denoted "$\Rightarrow$") the union of the images of $\beta(Q_1)$ under these mappings. □

We might be tempted to generalize the algorithm CQstable to test the stability of a CQAC. However, the following example from [Ull89] shows that a CQAC may not have a minimal equivalent that has a subset of its subgoals.

EXAMPLE 5.1 Consider the query

$$ans(X, Y) \text{ :- } p(X, Y), X \neq Y, X \leq Y$$

Clearly the query is not equivalent to the query formed from any subset of its subgoals, However,

$$ans(X, Y) \text{ :- } p(X, Y), X < Y$$

is an equivalent query with fewer subgoals. □

[3] This theorem is a variation of the results in [GSUW94, Klu88]. In particular, [GSUW94] assumes that no variable appears twice among their ordinary subgoals, and no constant appears in their ordinary subgoals.

As another example, the following query

$$Q : ans(Y) \text{ :- } p(X), r(X,Y), r(A,B), A < B, X \leq A, A \leq Y, X \leq Y$$

is equivalent to query

$$Q' : ans(Y) \text{ :- } p(X), r(X,Y), r(A,B), A < B, X \leq A, A \leq Y$$

In particular, comparison $X \leq Y$ in Q is redundant and can be removed to make Q minimal. As we will see shortly, a simple extension of the CQstable* algorithm cannot test the stability of the minimal query Q' either.

5.1 Answerable Subquery of a CQAC

Definition 5.1 (**answerable subquery of a CQAC**) Let Q be a CQAC on relations with binding restrictions. Its *answerable subquery*, denoted by Q_a, is the query that has the head of Q, the answerable subgoals of Q, and all the comparisons of the bound variables that can be derived from $\beta(Q)$. □

The answerable subquery Q_a of a CQAC Q can be computed as follows. First using the binding restrictions of the relations, compute all the answerable ordinary subgoals $\mathcal{A}(Q)$ of query Q using the algorithm Inflationary. Let $\mathcal{V}$ be the set of all the bound variables in $\mathcal{A}(Q)$. Derive all the inequalities $\mathcal{I}$ among the variables in $\mathcal{V}$ from $\beta(Q)$. Query Q_a includes *all* the constraints of $\mathcal{V}$, because $\beta(Q)$ may derive more constraints that $\mathcal{V}$ should satisfy. For instance, assume X is a variable in Q_a, and variable Y is not. If $\beta(Q)$ has comparisons $X < Y$ and $Y < 5$, then variable X in Q_a still needs to satisfy the constraint $X < 5$.

We might want to generalize the algorithm CQstable* as follows. Given a CQAC Q, we compute its answerable subquery Q_a. We test the stability of Q by testing whether $Q_a \sqsubseteq Q$, which can be tested using the algorithms in [GSUW94, ZO93] ("the GZO algorithm" for short). However, the following example shows that this "algorithm" does not always work.

EXAMPLE 5.2 Consider query

$$Q : ans(Y) \text{ :- } p(X), r(X,Y), r(A,B), A < B, X \leq A, A \leq Y$$

where relation p has a binding pattern *f*, and relation r has a binding pattern *bf*. In the first step, we find all the answerable subgoals $p(X)$ and $r(X,Y)$ of query Q. With variables X and Y bound, we derive all the possible constraints these two variables must satisfy from the comparisons in Q. The only derived comparison is $X \leq Y$. Thus we get the answerable subquery:

$$Q_a : ans(Y) \text{ :- } p(X), r(X,Y), X \leq Y$$

Using the GZO algorithm we know that $Q_a \not\sqsubseteq Q$. Therefore, we may claim that query Q is not stable. However, query Q *is* stable. As we will see in Section 5.3, query Q is equivalent to the union of the following two queries.

$$\begin{array}{ll} T_1: & ans(Y) \text{ :- } p(X), r(X,Y), X < Y \\ T_2: & ans(Y) \text{ :- } p(Y), r(Y,Y), r(Y,B), Y < B \end{array}$$

Note both T_1 and T_2 are stable, since all their ordinary subgoals are answerable. $\Box$

The above testing procedure gives the wrong claim about the query's stability because the only case where $Q_a \not\sqsubseteq Q$ is when $X = Y$. However, comparisons $X \leq A$ and $A \leq Y$ will then force A to be equal to X and Y, and the subgoal $r(A, B)$ becomes answerable. In general, one subtle point is that "$\leq$" and "$\geq$" may imply some equality constraints, which may make some subgoals feasible. This example suggests that we need to consider *all* the total orders of the variables in a CQAC query to test its stability. Formally, a total order of the variables in the query is an order with some equalities, i.e., all the variables are partitioned to sets $S_1, \ldots, S_k$, such that each S_i is a set of equal variables, and for any two variables $X_i \in S_i$ and $X_j \in S_j$, if $i < j$, then $X_i < X_j$.

5.2 Algorithm: CQAC1stable

Before giving an algorithm for testing the stability of any CQAC, we first consider the case where the above testing procedure works. It turns out that the procedure is correct when a CQAC does not include comparisons $\{\leq, \geq\}$, i.e., their comparisons only have $\{<, >, \neq\}$. Figure 8 shows the corresponding algorithm *CQAC1stable*.

Algorithm CQAC1stable: Test stability of CQACs with comparisons $\{<, >, \neq\}$ only.
Input: • Q: A CQAC with comparisons $\{<, >, \neq\}$.
• B: Binding patterns of the relations used in Q.
Output: Decision about the stability of Q.
Method:
(1) Compute the answerable subquery Q_a of Q:
- Based on B, compute all the answerable ordinary subgoals $\mathcal{A}$ using the algorithm Inflationary.
- Derive all the inequalities $\mathcal{I}$ among the bound variables in $\mathcal{A}$ from $\beta(Q)$.
- Let Q_a be the query with $\mathcal{A}$, $\mathcal{I}$, and the head of Q.

(2) Test whether $Q_a \sqsubseteq Q$ using the GZO algorithm.
(3) If $Q_a \sqsubseteq Q$, then Q is stable; otherwise, Q is not stable.

Figure 8: Algorithm: CQAC1stable

Theorem 5.2 *The algorithm CQAC1stable correctly decides the stability of a CQAC with comparisons* $\{<, >, \neq\}$ *only.* $\Box$

$$\begin{array}{ll}
Q : H \text{ :- } O_1, \ldots, O_k, \ldots, O_n, C_1, \ldots, C_m & \Leftarrow \text{instantiation } f \text{ (database } D_2) \\
 & \qquad\qquad\quad \big\uparrow \text{extend} \\
Q_a : H \text{ :- } O_1, \ldots, O_k, C_1', \ldots, C_m' & \Leftarrow \text{instantiation } s \text{ (database } D_1)
\end{array}$$

Figure 9: Correctness proof of algorithm CQAC1stable

Proof: If query Q_a is equivalent to query Q, then Q is stable, since for any database D, we can compute $ANS(Q, D)$ by computing $ANS(Q_a, D)$, which is computable since all the ordinary subgoals of Q_a are answerable.

Now, we prove that if $Q_a \not\sqsubseteq Q$, query Q cannot be stable. We need to construct two databases D_1 and D_2, such that $ANS(Q, D_1) \neq ANS(Q, D_2)$, but these two databases have the same observable tuples. Figure 9 shows the main idea of the construction. Assume that Q have n ordinary subgoals, $O_1, \ldots, O_n$. Without loss of generality, let $O_1, \ldots, O_k$ be the answerable subgoals of Q. These k subgoals are also all the ordinary subgoals of Q_a. Let $\mu_1, \ldots, \mu_k$ be all the containment mappings from $O(Q)$ to $O(Q_a)$. Since $Q_a \not\sqsubseteq Q$, by Theorem 5.1, $\beta(Q_a)$ does not imply $\mu_1(\beta(Q)) \vee \ldots \vee \mu_k(\beta(Q))$. Then there must be an instantiation s for the variables in Q_a, such that $s(\beta(Q_a))$ is true, while no μ_i can make $s\Big(\mu_i(\beta(Q))\Big)$ true. Notice since $\beta(Q)$ does not imply equalities, different variables are assigned different constants under s.

Let database D_1 include tuples $t_1, \ldots, t_k$ under the instantiation s. Notice that: (1) Q_a only has comparisons $\{<, >, \neq\}$, and (2) the comparisons in $\beta(Q_a)$ are *all* the inequality constraints that the variables in Q_a should satisfy. Therefore, we can always extend the instantiation s to an instantiation f of the variables in Q, by assigning new distinct values to the unbound variables in Q.[4] This instantiation f also turns the head of Q_a to a tuple t_h. Let database D_2 include the tuples of Q under the instantiation f.

Since instantiation f uses new distinct values for the unbound variables in Q, the tuples for the nonanswerable subgoals of Q_a cannot be retrieved under D_2, given the binding restrictions of the relations. Therefore, tuples $t_1, \ldots, t_k$ are all the observable tuples under both databases D_1 and D_2. We only need to prove that $t_h \notin ANS(Q, D_1)$. Otherwise we can construct a containment mapping μ from $O(Q)$ to $O(Q_a)$, such that $s\Big(\mu(\beta(Q))\Big)$ is true, contradicting to the fact that no μ_i can make $s\Big(\mu_i(\beta(Q))\Big)$ true. ∎

The algorithm CQAC1stable also shows how to compute the complete answer to a stable CQAC Q with comparisons $\{<, >, \neq\}$ only. For any database D, we compute $ANS(Q, D)$ by computing $ANS(Q_a, D)$. That is, we first use a linear plan following a feasible order of the subgoals $O(Q_a)$ to solve these subgoals. Then we filter out the tuples in the corresponding supplementary relation that do not satisfy the comparisons in $\beta(Q_a)$.

EXAMPLE 5.3 Consider the following query with $\{<, >\}$ comparisons only.

$$Q : ans(B) \text{ :- } p(B), r(A, B), r(A, C), A < C, C < B$$

Relation p has a binding pattern *f*, and relation r has a binding pattern *fb*. We use the algorithm CQAC1stable to test its stability. Clearly subgoals $p(B)$ and $r(A, B)$ are answerable and the bound variables are A and B. We derive all the inequalities of A and B from $A < C$ and $C < B$. The only derived inequality is $A < B$. Thus the following is the answerable subquery of Q:

$$Q_a : ans(B) \text{ :- } p(B), r(A, B), A < B$$

We then test whether $Q_a \sqsubseteq Q$. There is only one containment mapping μ from $O(Q)$ to $O(Q_a)$:

$$\mu(B) = B; \mu(A) = A; \mu(C) = B.$$

We verify whether $\beta(Q_a)$ logically implies $\mu(\beta(Q))$:

$$A < B \Rightarrow A < B \wedge B < B$$

which is false, and we have $Q_a \not\sqsubseteq Q$. Therefore, query Q is not stable.

[4]This extension may not be possible if conditions (1) and (2) are not satisfied. Example 5.2 is an example.

The nonstability of query Q can also be proved by the following two databases:

$$D_1 = \{p(3), r(1,3), r(1,2)\}, D_2 = \{p(3), r(1,3)\}.$$

Clearly $ANS(Q, D_1) = \{3\}$, and $ANS(Q, D_2) = \phi$, but these two databases have the same observable tuples $\{p(3), r(1,3)\}$. Note that tuple $r(1,2)$ cannot be retrieved because "2" can represent any constant in range $(1,3)$. $\square$

Note that if we replace "<" with "$\leq$" in the above example, then query Q becomes stable.

5.3 Algorithm: CQACstable

Now we show how to test stability of general CQACs by giving the following theorem.

Theorem 5.3 *Let Q be a CQAC, and $\Omega(Q)$ be the set of all the total orders of the variables in Q that satisfy the comparisons of Q. For each total order $\lambda \in \Omega(Q)$, let Q^λ be the corresponding query that includes the ordinary subgoals of Q and all the inequalities and equalities of this order λ. Equivalent arguments in Q^λ are substituted by one argument of them. Then query Q is stable if and only if for all $\lambda \in \Omega(Q)$, query $Q_a^\lambda \sqsubseteq Q$, where Q_a^λ is the answerable subquery of Q^λ.* $\square$

$$Q \equiv \bigcup \begin{cases} Q^{\lambda_1} & \sqsubseteq & Q_a^{\lambda_1} & \sqsubseteq & Q \\ Q^{\lambda_2} & \sqsubseteq & Q_a^{\lambda_2} & \sqsubseteq & Q \\ \vdots & & & & \\ Q^{\lambda_m} & \sqsubseteq & Q_a^{\lambda_m} & \sqsubseteq & Q \end{cases}$$

Figure 10: Proof of Theorem 5.3, "*If*" part

Proof: *If*: Assume that for each $\lambda_i \in \Omega(Q)$, $Q_a^{\lambda_i} \sqsubseteq Q$. As shown in Figure 10, $Q \equiv \bigcup_{\lambda \in \Omega(Q)} Q^\lambda$. In addition, for each $\lambda_i \in \Omega(Q)$, $Q^{\lambda_i} \sqsubseteq Q_a^{\lambda_i}$. Then we have

$$Q \equiv \bigcup_{\lambda \in \Omega(Q)} Q^\lambda \sqsubseteq \bigcup_{\lambda \in \Omega(Q)} Q_a^\lambda \sqsubseteq Q.$$

Thus $Q \equiv \bigcup_{\lambda \in \Omega(Q)} Q_a^\lambda$. For any database D, we can compute $ANS(Q, D)$ by computing $ANS(Q_a^\lambda, D)$ for each total order $\lambda \in \Omega(Q)$. This answer is computable since all its subgoals are answerable. Then we take the union of these answers as $ANS(Q, D)$. Therefore, query Q is stable.

Only If: Assume there is a total order in $\Omega(Q)$, say λ_1, such that $Q_a^{\lambda_1} \not\sqsubseteq Q$. For instance, Figure 11 shows an example of a total order. The variables on the left-hand side must be smaller than the variables on the right-hand side. Some variables might be equal to each other. For instance, the variables in the figure must satisfy:

$$A_1 < A_2 = A_3 < A_4 < A_5 = A_6 = A_7 < A_8 < A_9 < A_{10} = A_{11}$$

Some variables (e.g., A_1, A_4, A_5, A_6, A_7, and A_9) are bound by the answerable subgoals.

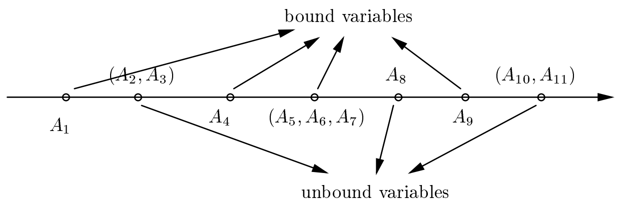

Figure 11: A total order

Now we prove query Q cannot be stable. We need to construct two databases D_1 and D_2, such that $ANS(Q, D_1) \neq ANS(Q, D_2)$, but D_1 and D_2 have the same observable tuples. The construction is essentially the same as the construction in the proof of Theorem 5.2. That is, let s be the instantiation of the variables in $Q_a^{\lambda_1}$ that makes $\beta(Q_a^{\lambda_1}) \Rightarrow \beta(Q)$ false. These variables correspond to the bound variables in the total order λ_1. Let D_1 include the tuples under the instantiation s.

Since $Q_a^{\lambda_1} \not\sqsubseteq Q$, query $Q_a^{\lambda_1}$ must have fewer subgoals than Q, and some subgoals in Q are not answerable. In addition, some variables are not bound. For those unbound variables, we can choose *new distinct* values for them. That is, the unbound variables and the bound variables have different values. Therefore, we can always extend instantiation s to a new instantiation f for the variables in Q, such that f uses new distinct values for the unbound variables. Let D_2 include the tuples corresponding to the instantiation f. By the construction of D_1 and D_2, they have the same observable tuples (i.e., the tuples in D_1). Following the same idea in the proof of Theorem 5.2, we can prove that $ANS(Q, D_1)$ does not include the tuple $t_h = f(G)$, where G is the head of Q. Therefore, query Q is not stable. ∎

Theorem 5.3 gives an algorithm *CQACstable* (shown in Figure 12) that tests the stability of any CQAC.

Algorithm CQACstable: Test stability of CQACs.
Input: • Q: A conjunctive query with arithmetic comparisons.
• B: Binding patterns of the relations used in Q.
Output: Decision about the stability of Q.
Method:
(1) Compute all the total orders $\Omega(Q)$ of the variables in Q that satisfy the comparisons in Q.
(2) For each $\lambda \in \Omega(Q)$:
- Let Q^λ be the corresponding query, where equivalent arguments are substituted by one argument;
- Compute the answerable subquery Q_a^λ of query Q^λ;
- Test $Q_a^\lambda \sqsubseteq Q$ by calling the GZO algorithm;
- If $Q_a^\lambda \not\sqsubseteq Q$, then query Q is not stable.

(3) Claim that query Q is stable.

Figure 12: Algorithm: CQACstable

The algorithm CQACstable also shows how to compute the complete answer to a stable CQAC Q for a database D. That is, let $Q = \bigcup_{\lambda \in \Omega(Q)} Q^\lambda$. For each query Q^λ, we compute $ANS(Q_a^\lambda, D)$, where Q_a^λ is the answerable subquery of Q^λ. Since Q_a^λ has a feasible order of all its ordinary subgoals, we compute $ANS(Q_a^\lambda, D)$ by using a linear plan following this order, and filtering out the results using the comparisons in Q_a^λ. We take the union of the answers for all the total orders in $\Omega(Q)$ as $ANS(Q, D)$.

EXAMPLE 5.4 Consider the query Q in Example 5.2. It has the following 8 total orders:

$$\begin{array}{ll} \lambda_1 : X < A = Y < B & \lambda_5 : X = A = Y < B \\ \lambda_2 : X < A < Y < B & \lambda_6 : X = A < Y < B \\ \lambda_3 : X < A < Y = B & \lambda_7 : X = A < Y = B \\ \lambda_4 : X < A < B < Y & \lambda_8 : X = A < B < Y \end{array}$$

For each of total order λ_i, we can write its corresponding query Q^{λ_i}. Here are all the 8 queries.

$$\begin{array}{ll} Q^{\lambda_1}: & ans(Y) \text{ :- } p(X), r(X,Y), r(Y,B), X < Y, Y < B \\ Q^{\lambda_2}: & ans(Y) \text{ :- } p(X), r(X,Y), r(A,B), X < A, A < Y, Y < B \\ Q^{\lambda_3}: & ans(Y) \text{ :- } p(X), r(X,Y), r(A,Y), X < A, A < Y \\ Q^{\lambda_4}: & ans(Y) \text{ :- } p(X), r(X,Y), r(A,B), X < A, A < B, B < Y \\ Q^{\lambda_5}: & ans(Y) \text{ :- } p(Y), r(Y,Y), r(Y,B), Y < B \\ Q^{\lambda_6}: & ans(Y) \text{ :- } p(X), r(X,Y), r(X,B), X < Y, Y < B \\ Q^{\lambda_7}: & ans(Y) \text{ :- } p(X), r(X,Y), r(X,Y), X < Y \\ Q^{\lambda_8}: & ans(Y) \text{ :- } p(X), r(X,Y), r(X,B), X < B, B < Y \end{array}$$

For each total order λ_i, we construct its corresponding answerable subquery $Q_a^{\lambda_i}$. The following are the 8 answerable subqueries.

$$\begin{array}{ll} Q_a^{\lambda_1}: & ans(Y) \text{ :- } p(X), r(X,Y), r(Y,B), X < Y, Y < B \\ Q_a^{\lambda_2}: & ans(Y) \text{ :- } p(X), r(X,Y), X < Y \\ Q_a^{\lambda_3}: & ans(Y) \text{ :- } p(X), r(X,Y), X < Y \\ Q_a^{\lambda_4}: & ans(Y) \text{ :- } p(X), r(X,Y), X < Y \\ Q_a^{\lambda_5}: & ans(Y) \text{ :- } p(Y), r(Y,Y), r(Y,B), Y < B \\ Q_a^{\lambda_6}: & ans(Y) \text{ :- } p(X), r(X,Y), r(X,B), X < Y, Y < B \\ Q_a^{\lambda_7}: & ans(Y) \text{ :- } p(X), r(X,Y), X < Y \\ Q_a^{\lambda_8}: & ans(Y) \text{ :- } p(X), r(X,Y), r(X,B), X < B, B < Y \end{array}$$

Each of the 8 answerable subquery can be proved to be contained in Q. Therefore, query Q is stable. Actually, if we combine queries Q^{λ_1}, $Q_a^{\lambda_2}$, Q^{λ_3}, Q^{λ_4}, Q^{λ_6}, $Q_a^{\lambda_7}$, $Q_a^{\lambda_8}$, and we have query:

$$Q^{\lambda_{1,2,3,4,6,7,8}} : ans(Y) \text{ :- } p(X), r(X,Y), r(A,B), X < Y, A < B, X \leq A, A \leq Y$$

and its answerable subquery is:

$$Q_a^{\lambda_{1,2,3,4,6,7,8}} : ans(Y) \text{ :- } p(X), r(X,Y), X < Y$$

which is the query T_1 in Example 5.2. We can prove

$$T_1 = Q_a^{\lambda_{1,2,3,4,6,7,8}} \equiv Q^{\lambda_{1,2,3,4,6,7,8}}.$$

In addition, Q^{λ_5} is equivalent to the query T_2 in the example. Since query $Q \equiv Q^{\lambda_{1,2,3,4,6,7,8}} \cup Q^{\lambda_5}$, we have $Q \equiv T_1 \cup T_2$. □

6 Stability of Datalog Queries

In this section we study stability of datalog queries, i.e., Horn-clause programs without function symbols [Ull89]. We first give a sufficient condition for stability of datalog queries, and then prove that testing stability of a datalog program is not decidable.

EXAMPLE 6.1 Suppose $flight$ is a finite relation with a binding adornment *bf*, and $flight(F, T)$ means that there is a nonstop flight from airport F to airport T. An IDB relation *reachable* is defined by the following two rules:

$$\begin{array}{ll} r_1: & reachable(X,Y) \text{ :- } flight(X,Y) \\ r_2: & reachable(X,Y) \text{ :- } flight(X,Z), reachable(Z,Y) \end{array}$$

Let queries Q_1 and Q_2 be *reachable*(sfo, X) and *reachable*(X, sfo), respectively. That is, query Q_1 asks for all the airports that are reachable from the airport *sfo*, while query Q_2 asks for all the airports from which the airport *sfo* is reachable. Although we cannot retrieve all the flight facts, the answer to query Q_1 can still be computed as follows: we query the relation to retrieve all the airports that are reachable from *sfo* via one nonstop flight. For each of these airports, we query the relation to retrieve its one-nonstop reachable airports. We repeat the process until no new airports are found. This process will terminate, since the $flight$ relation is finite. The set of the retrieved airports is the complete answer to query Q_1. The reason is that for any airport a in the answer, there exists a chain of distinct airports $a_1 = sfo, a_2, \ldots, a_n = a$, such that for $i = 1, \ldots, n-1$, $\langle a_i, a_{i+1} \rangle$ is a tuple in the $flight$ relation. Therefore, airport a_i can be retrieved in the $(i-1)$st step during the computation.

For query Q_2, we cannot compute its answer in the same way as Q_1, because the $flight$ relation does not allow us to retrieve its facts in a "forward" way. In fact, we cannot know the complete answer to query Q_2 at all, since there can always be an airport from which sfo is reachable, but this airport cannot be retrieved from the relation. □

6.1 A Sufficient Condition for Datalog Stability

We first give a sufficient condition for stability of datalog queries. We assume that readers are familiar with the concept of rule/goal graphs described in Chapter 12 in [Ull89]. Here we give a brief overview. Given a set of rules and a query goal p^α with an adornment α (a string of *b*'s and *f*'s), a rule/goal graph (RGG for short) indicates the order in which subgoals are to be evaluated in these rules, and indicates the way in which variable bindings pass from one subgoal to another within a rule.

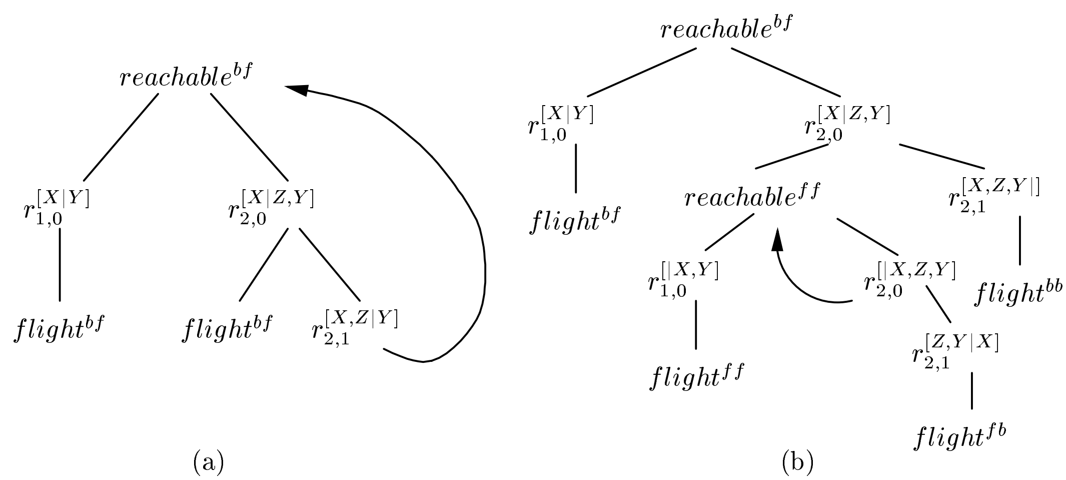

Figure 13: Two RGGs for the query goal *reachable*^{bf}

EXAMPLE 6.2 Consider the query Q_1 in Example 6.1. The query can be represented as a goal $reachable^{bf}$, i.e., the problem of determining, given a fixed value (i.e., *sfo*) for the first argument, the set of Y such that $reachable(sfo, Y)$ is true. Figure 13(a) shows an RGG of the goal. In the graph, there are two different kinds of nodes: goal nodes and rule nodes. A goal node is a predicate with a binding adornment that specifies which arguments are bound and which are not. For instance, the root node $reachable^{bf}$ is a goal node indicating that the first argument of predicate *reachable* has been bound, and the second argument is not.

A rule node indicates the binding status of the variables in a rule. Its subscript indicates the stage of processing the subgoals in the rule from left to right. Its superscript specifies what variables have been bound so far, either by a previous subgoal, or by the head of this rule. The superscript also specifies what variables have not been bound. For instance, the node $r_{1,0}^{[X|Y]}$ is a rule node. Its subscript "$1,0$" means that it corresponds to the stage when no subgoal of rule r_1 has been processed. Its superscript "$[X|Y]$" means that at this stage, variable X is bound (by the head of the rule) and variable Y is not. Similarly, the rule node $r_{2,0}^{[X|Z,Y]}$ specifies that when no subgoal of rule r_2 has been processed, variable X is bound, and variables Z and Y are not. The rule node $r_{2,1}^{[X,Z|Y]}$ means that after the first subgoal of rule r_2 is processed, variables X and Z are bound, and Y is not. The RGG in Figure 13(a) is constructed respecting the order in which the subgoals are written. For a different order we may have a different RGG. For instance, if we switch the two subgoals in rule r_2, we will have a new RGG, as shown in Figure 13(b). □

We assume that a set of rules has the *unique binding pattern* property with respect to a given adorned goal. That is, when we construct the RGG starting with the adorned goal and following the order of the subgoals of the rules as written, no IDB predicate appears with two different adornments. If a set of rules P does not have this property with respect to a query goal, we can call the Algorithm 12.7 in [Ull89] to rewrite these rules and the goal, and generate a revised set of rules P' that has the unique binding pattern property with respect to the query goal. We can show that if the query goal in P' is feasible with respect to the relations' binding patterns (defined shortly) iff the query goal in P is feasible.

Given a set of rules on EDB relations with binding restrictions, an RGG of a goal node p^{α} is *feasible* if all its EDB goals in the RGG use only the adornments that are permitted by the EDB relations. For instance, the RGG in Figure 13(a) is a feasible RGG, since all the EDB goals in the graph (the two nodes of $flight^{bf}$) are permitted by the $flight$ relation. However, the RGG in Figure 13(b) is not feasible, since it has two EDB goals ($flight^{ff}$ and $flight^{fb}$) that are not permitted by the $flight$ relation.

Theorem 6.1 *If a set of rules on EDB relations with binding restrictions has a feasible RGG with respect to a query goal, then the query is stable.* □

Proof: Assume that the set of rules and the query goal p^{α} have a feasible RGG. We apply the magic-sets transformation (as described in [Ull89, Chapter 13]) on the rules and the goal, and get a new set of rules $\mathcal{R}$ such that the relation for p is the answer to the query. For any instance of the EDB relations, consider the case where the relations did not have restrictions. The complete answer to the query p can be computed using a bottom-up evaluation of the rules $\mathcal{R}$.

Since the EDB relations do have binding restrictions, we should consider whether the bottom-up evaluation of $\mathcal{R}$ can be executed. Because G is a feasible RGG, during the bottom-up evaluation of

$\mathcal{R}$, each time an EDB subgoal is evaluated, we have the necessary bindings to query the relation. Therefore, we can still execute the bottom-up evaluation. In addition, in each step of the evaluation, the facts we need to compute are the same as the facts we computed in the bottom-up evaluation if the EDB relations did not have any restrictions. Therefore, we can compute the complete answer to the query using a bottom-up evaluation of $\mathcal{R}$. ∎

The proof of Theorem 6.1 also gives an algorithm for computing the complete answer to a query goal if it has a feasible RGG. That is, we apply the magic-sets transformation to the rules and the goal to get a set of rules $\mathcal{R}$. We evaluate these rules using a bottom-up evaluation. In each step, we evaluate a rule following the order in which the RGG is constructed. By the construction of the rules $\mathcal{R}$, each time we solve an EDB subgoal, we have enough bindings to evaluate this subgoal.

[Mor88] gave an algorithm for testing the existence of a feasible RGG given a set of rules and a query goal. The algorithm is inherently exponential in time. However, if there is a bound on the arity of predicates, then the algorithm with this heuristic takes polynomial time [UV88].

6.2 Undecidability Result

In some cases, even though a set of rules does not has a feasible RGG with respect to a query goal, the query may still be stable, since we may rewrite the rules to obtain a new set of rules that has a feasible RGG with respect to the query goal.

For example, if we add another subgoal $flight(W, Z)$ to rule r_2, then the new set of rules does not have a feasible RGG with respect to the query goal of Q_1 , since the variable W in the third subgoal $flight(W, Z)$ cannot be bound. However, the new query is equivalent to the old query, and query Q_1 on the new rules is still stable. This example shows a similar phenomenon as CQs; that is, we need to "minimize" datalog rules before checking the existence of a feasible RGG. However, minimizing datalog rules is much harder than minimizing CQs and UCQs. Shmueli [Shm93] showed that for a datalog program Q, whether Q is equivalent to datalog program Q', where Q' is produced by removing a subgoal from a rule of Q, is undecidable. Not surprisingly, we have the following result.

Theorem 6.2 *Stability of datalog programs is undecidable.* □

Proof: Let Q_1 and Q_2 be two arbitrary datalog queries. We show that a decision procedure for the stability of datalog programs would allow us to decide whether $Q_1 \sqsubseteq Q_2$. Since containment of datalog programs is undecidable, we prove the claim.[5] Let all the EDB relations in the two queries have an all-free binding pattern; i.e., there is no restriction of retrieving tuples from these relations. Without loss of generality, we can assume that the goal predicates in Q_1 and Q_2, named p_1 and p_2 respectively, have arity m. Let $\mathcal{Q}$ be the datalog query consisting of all the rules in Q_1 and Q_2, and of the rules:

$$\begin{array}{lll} r_1: & ans(X_1, \ldots, X_m) & \text{:- } p_1(X_1, \ldots, X_m), e(Z) \\ r_2: & ans(X_1, \ldots, X_m) & \text{:- } p_2(X_1, \ldots, X_m) \end{array}$$

where e is a new 1-ary relation with the binding pattern b, and Z is a new argument that does not appear in $X_1, \ldots, X_m$. We show that $Q_1 \sqsubseteq Q_2$ if and only if query $\mathcal{Q}$ is stable.

[5]The idea of the proof is borrowed from [Dus97], Chapter 2.3.

"Only If": Assume $Q_1 \sqsubseteq Q_2$. Hence $\mathcal{Q} \equiv Q_2$. Since the EDB relations in Q_2 can return all their tuples for free, Q_2 (thus $\mathcal{Q}$) is stable.

"If": Assume $Q_1 \not\sqsubseteq Q_2$, we prove that query $\mathcal{Q}$ cannot be stable. Since Q_1 is not contained in Q_2, there exists a database D of the EDB relations in Q_1 and Q_2, such that $ANS(Q_1, D) \not\subseteq ANS(Q_2, D)$. That is, there is a tuple $t \in ANS(Q_1, D)$, while $t \notin ANS(Q_2, D)$. Now we construct two databases D_1 and D_2 of the EDB relations and the relation e, such that query $\mathcal{Q}$ has the same observable tuples under D_1 and D_2, but $ANS(\mathcal{Q}, D_1) \neq ANS(\mathcal{Q}, D_2)$.

Both D_1 and D_2 include D for the EDB relations in Q_1 and Q_2. However, in D_1, relation e is empty; in D_2, relation e has one tuple $\langle z \rangle$, while z is a new value that does not appear in any tuple in D. For both D_1 and D_2, the observable tuples are those in D, while we cannot get any tuple from relation e. Hence, rule r_1 cannot yield any answer to $\mathcal{Q}$, and the retrievable answer to $\mathcal{Q}$ is $ANS(Q_2, D)$ for both D_1 and D_2. For D_1, since relation e is empty, $ANS(\mathcal{Q}, D_1) = ANS(Q_2, D)$, which does not include tuple t. However, for D_2, relation e has a tuple $\langle z \rangle$, and $ANS(\mathcal{Q}, D_1) = ANS(Q_1, D) \cup ANS(Q_2, D)$, which includes tuple t. Therefore, $ANS(\mathcal{Q}, D_1) \neq ANS(\mathcal{Q}, D_2)$, and query $\mathcal{Q}$ is not stable. ∎

7 Nonstable Queries with Computable Complete Answers

If a query is not stable, in some cases, we may still compute its complete answer, and the computability is data dependent. In this section we discuss in what cases the complete answer to a nonstable conjunctive query may be computed. We develop a decision tree that guides the planning process to compute the complete answer to a CQ. We discuss two planning strategies that can be taken while traversing the tree. We also discuss how to optimize a CQ to compute its complete answer.

7.1 Dynamic Computability of Complete Answers

In [LC00, LC01a] we show that we can compute an answer to a query by borrowing bindings from relations not in the query, even though the borrowed bindings do not guarantee to retrieve tuples from relations. For instance, suppose a Web search form requires a book title to return book information. We can go to other sources (e.g., The Library of Congress, http://www.loc.gov/) to retrieve titles, and then use these titles to access the Web site. These papers discuss how to use an *exhaustive plan* to retrieve as much information as possible from relations, and use the information to compute the maximal answer to a query. The papers also discussed how to trim irrelevant relations that do not help in obtaining bindings.

The following example shows that using an exhaustive plan we may compute the complete answer to a nonstable CQ in some cases, and we do not know the computability until some plan is executed.

EXAMPLE 7.1 Suppose that we have a relation $r(A, B, C)$ with one binding pattern *bff*, a relation $s(C, D)$ with a binding pattern *fb*, and a relation $p(D, E)$ with a binding pattern *ff*. The attributes $A, B, \ldots, E$ are from different domains. Consider the following two queries.

$$Q_1: \quad ans(B) \text{ :- } r(a, B, C), s(C, D).$$
$$Q_2: \quad ans(D) \text{ :- } r(a, B, C), s(C, D).$$

The two queries have the same subgoals but different heads. They are not stable, since their minimal equivalents (themselves) are not feasible. However, we can still try to answer query Q_1 as follows. Send a query $r(a, X, Y)$ to relation r. Assume this access returns three tuples: $\langle a, b_1, c_1\rangle$, $\langle a, b_2, c_2\rangle$, and $\langle a, b_2, c_3\rangle$. The supplementary relation I_1 after the first subgoal has the schema BC, and contains three tuples: $\langle b_1, c_1\rangle$, $\langle b_2, c_2\rangle$, and $\langle b_2, c_3\rangle$. Although we cannot use the new bindings $\{c_1, c_2, c_3\}$ for attribute C to query relation s directly due to its *fb* binding pattern, we may still use an exhaustive plan to retrieve tuples from relation s, e.g., using the D bindings provided by relation p, although p is not mentioned in the query.

If the exhaustive plan retrieves two tuples $\langle c_1, d_1\rangle$ and $\langle c_2, d_2\rangle$ from relation s, we can still claim that the complete answer is $\{b_1, b_2\}$. The reason is that the only distinguished variable B is bound by the supplementary relation I_1, and we are able to obtain all correct B values using goals from the query. In particular, tuples $\langle b_1, c_1\rangle$, $\langle b_2, c_2\rangle$, and $\langle b_2, c_3\rangle$ are the only tuples in I_1, and their projection onto variable B is $\{b_1, b_2\}$. Tuples $\langle b_1, c_1\rangle$ and $\langle b_2, c_2\rangle$ in I_1 can join with tuples $\langle c_1, d_1\rangle$ and $\langle c_2, d_2\rangle$ in s, respectively, and their projection onto the variable B is also $\{b_1, b_2\}$. Thus, this projection is the complete answer. On the other hand, if the exhaustive plan retrieves only one tuple $\langle c_2, d_2\rangle$ from relation s, then we do not know whether $\langle b_2\rangle$ is the complete answer or not, since we do not know whether relation s has a tuple $\langle c_1, d\rangle$ (for some d value) that has not been retrieved. This tuple can join with the tuple $\langle b_1, c_1\rangle$ in I_1 to produce the value b_1 as an answer.

We can also try to answer query Q_2 in the same way. After the first subgoal is solved, the supplementary relation I_1 includes three tuples $\langle b_1, c_1\rangle$, $\langle b_2, c_2\rangle$, and $\langle b_2, c_3\rangle$. However, even if an exhaustive plan is executed to retrieve tuples from relation s, we can never know the complete answer to Q_2, since there can always be a tuple $\langle c_1, d'\rangle$ in relation s that has not been retrieved, and d' is in the complete answer to Q_2. In other words, we are not able to obtain the correct values for the distinguished variable D from goals in the query. Even if we resort to external relations, we still have no guarantee that they provide all the good values.

For both queries Q_1 and Q_2, if the supplementary relation I_1 is empty, then we can claim that their complete answers are both empty. □

An important observation on the two queries is that in query Q_1, the distinguished variable B can be bound by the answerable subgoal $r(a, B, C)$, while in query Q_2, the distinguished variable D cannot be bound by the answerable subgoal $r(a, B, C)$. In general, if a minimal CQ Q_m is not stable, we can use the algorithm Inflationary to find all its answerable subgoals Φ_a. If all the distinguished variables can be bound by Φ_a, i.e., the answerable subquery of Q_m is safe, then we use a linear plan of a feasible order of Φ_a to compute the supplementary relation (denoted I_a) of these subgoals. There are two cases:

1. If I_a is empty, then the complete answer to the query is empty.
2. If I_a is not empty, let I_a^P be the projection of I_a onto the distinguished variables. Execute an exhaustive plan to retrieve tuples for the nonanswerable subgoals Φ_{na}.
   (a) If for every tuple t^P in I_a^P, there is a tuple t_a in I_a, such that the projection of t_a onto the distinguished variables is t^P, and t_a can join with some tuples for all the subgoals Φ_{na} (tuple t^P is then called *satisfiable*), then I_a^P is the complete answer to the query.
   (b) Otherwise, we do not know the complete answer to the query.

If not all the distinguished variables are bound by the answerable subgoals Φ_a, i.e., the answerable subquery of Q_m is not safe, then the complete answer is not computable, unless the supplementary relation I_a is empty. The following lemmas prove the claims above.

Lemma 7.1 *For a minimal CQ Q_m, if the supplementary relation I_a of the answerable subgoals Φ_a is empty, then the complete answer to the query is empty.* □

Proof: Otherwise, if the complete answer has a tuple t, consider the tuples for the answerable subgoals Φ_a that contribute to the tuple t. By using a linear plan of a feasible order of Φ_a, we can retrieve these tuples. Therefore, the join of these tuples, with the values for the irrelevant attributes dropped, must be in the supplementary relation I_a, which cannot be empty. ■

Lemma 7.2 *Assume that Q_m is a minimal CQ, and all the distinguished variables are bound by the answerable subgoals Φ_a. Let I_a^P be the projection of I_a onto the distinguished variables. (1) If every tuple t^P in I_a^P is satisfiable, then I_a^P is the complete answer to the query. (2) Otherwise, the complete answer is not computable.* □

Proof: (1) Let t be a tuple in the complete answer, and suppose t comes from tuples $t_1, \dots, t_k$ of the answerable subgoals Φ_a and tuples $t_{k+1}, \dots, t_n$ of the nonanswerable subgoals Φ_{na}. Tuples $t_1, \dots, t_k$ must be retrieved by a linear plan of a feasible order of Φ_a during the computation of I_a. The projection of $t_1 \bowtie \cdots \bowtie t_k$ onto the distinguished variables is tuple t, since the distinguished variables are all bound by the subgoals Φ_a. Thus tuple t is in I_a^P. On the other hand, since every tuple t^P in I_a^P is satisfiable, t^P is in the answer to the query. Therefore, I_a^P is the complete answer.

(2) Let t^P be a tuple in I_a^P that is not satisfiable. Following the idea of the proof of Lemma 3.3, there can always be some tuples for the nonanswerable subgoals Φ_{na} that can join with a tuple in I_a that produces t^P, such that t^P is a tuple in the complete answer. However, these tuples for Φ_{na} cannot be retrieved because of the restrictions of Φ_{na}. Without these tuples, t^P is not in the complete answer. Since we do not know whether these tuples for Φ_{na} exist or not, we do not know whether the complete answer includes t^P or not. ■

Lemma 7.3 *For a minimal CQ Q_m, if not all the distinguished variables are bound by the answerable subgoals Φ_a, and the supplementary relation I_a is not empty, then the complete answer is not computable.* □

Proof: Let t_a be a tuple in I_a, and v be a distinguished variable that cannot be bound by Φ_a. Following the idea of the proof of Lemma 3.3, there can always be some tuples for the nonanswerable subgoals Φ_{na} that can join with the tuple t_a, such that the projection r of the join onto the distinguished variables (including v) is in the complete answer. However, these tuples cannot be retrieved because of the restrictions of Φ_{na}. Without these tuples, tuple r is not in the complete answer. Since we do not whether these tuples for Φ_{na} exist or not, we do not know whether the complete answer includes tuple r or not. ■

To summarize, whether the complete answer to a nonstable CQ is computable depends on the database, since it is not known until an exhaustive plan (Section 7.1) is executed.

7.2 The Decision Tree

We develop a decision tree (as shown in Figure 14) that guides the planning process to compute the complete answer to a CQ. The shaded nodes are where we can decide whether the complete answer is computable or not. Now we explain the decision tree in details. We first minimize a CQ Q by deleting its redundant subgoals, and compute its minimal equivalent Q_m (arc 1 in Figure 14). Then we test the feasibility of the query Q_m by calling the algorithm Inflationary. If it is feasible (arc 2 in Figure 14), Q_m (thus Q) is stable, and its answer can be computed by a linear plan following a feasible order of all the subgoals in Q_m.

If Q_m is not feasible (arc 3), we compute all its answerable subgoals Φ_a by calling the algorithm Inflationary, check if all the distinguished variables are bound by Φ_a. There are two cases:

1. If all the distinguished variables are bound by Φ_a (arc 4), then the complete answer may be computed even if the supplementary relation I_a of subgoals Φ_a is not empty. We compute I_a by a linear plan following a feasible order of Φ_a.

   (a) If I_a is empty (arc 5), then the complete answer is empty (Lemma 7.1).

   (b) If I_a is not empty (arc 6), we compute the relation I_a^P by projecting I_a onto the distinguished variables. We use an exhaustive plan to retrieve tuples for the nonanswerable subgoals Φ_{na}, and check whether all the tuples in I_a^P are satisfiable. If so (arc 7), then I_a^P is the complete answer. If not (arc 8), then the complete answer is not computable (Lemma 7.2).

2. If some distinguished variables are not bound by the subgoals Φ_a (arc 9), then the complete answer is not computable, unless the supplementary relation I_a is empty. Similarly to the case of arc 4, we compute I_a by a linear plan. If I_a is empty (arc 10), then the complete answer is empty. Otherwise (arc 11), the complete answer is not computable (Lemma 7.3).

While traversing the tree, if we reach a node where the complete answer is unknown, we still have some information about the lower bound and upper bound of the complete answer. For instance, if arc (8) is reached, then the upper bound of the answer is I_a^P (i.e., the answer can only be a subset of I_a^P), and the lower bound is all the satisfiable tuples in I_a^P. If arc (11) is reached, the answer has the lower bound ϕ, while it has no upper bound. In the shaded nodes where we can compute the complete answer, the lower bound and upper bound converge. We can tell the user the information about the lower bound and upper bound for decision support and analysis by the user.

Two strategies can be adopted while traversing the decision tree from the root to a leaf node, a pessimistic strategy and an optimistic strategy. In a node where we do not know whether the complete answer is computable until we traverse one level down the tree, a *pessimistic* strategy gives up traversing the tree. On the contrary, an *optimistic* strategy traverses one more level by doing the corresponding operations. What strategy should be taken is application dependent. For instance, we should consider how "eager" the user is for the complete answer to a query, how expensive a linear plan and an exhaustive plan are, how likely the supplementary relation I_a is to be empty, and how likely all the tuples in I_a^P are satisfiable. We may use statistics to answer these questions and make the decision about what strategy to take.

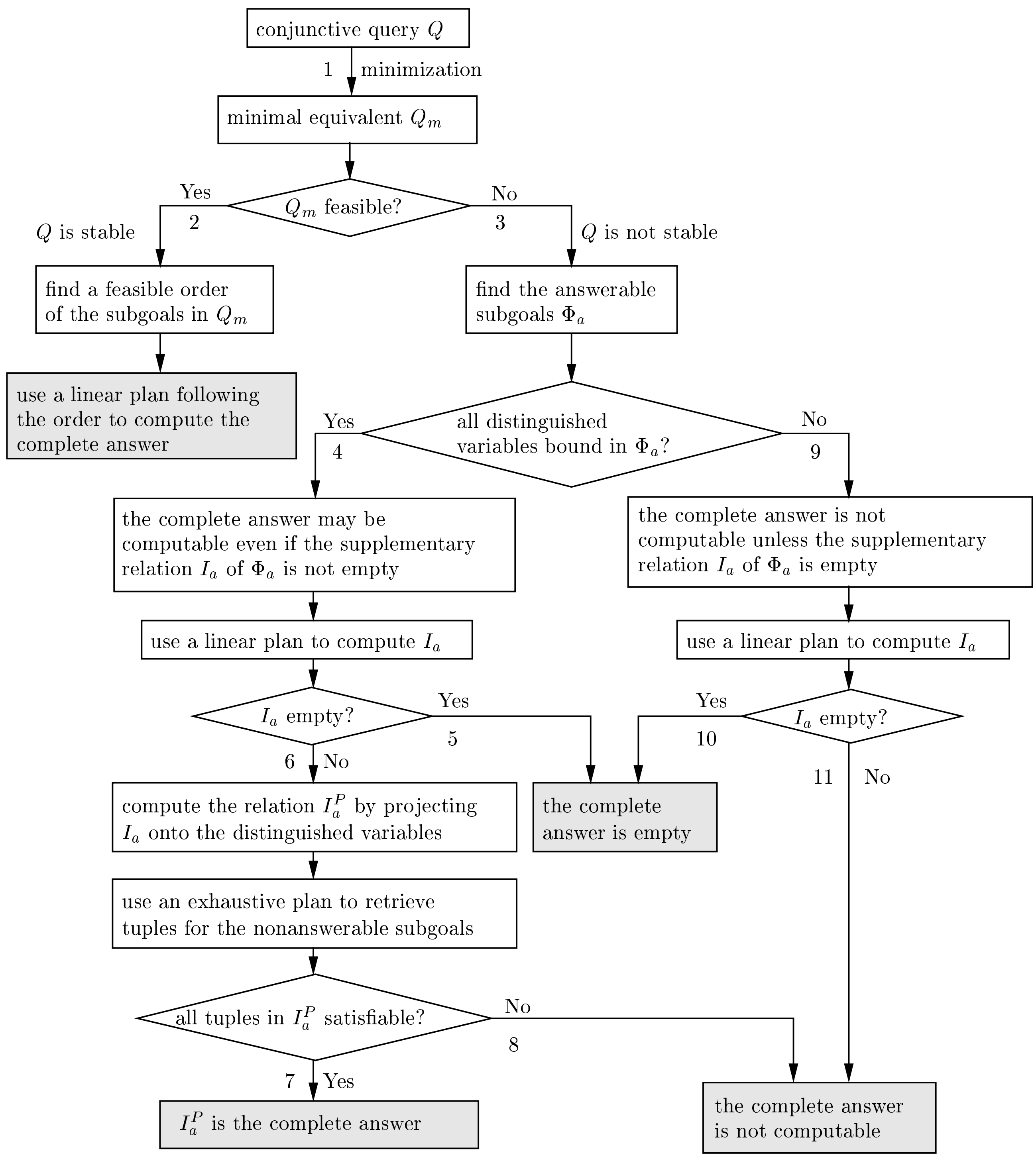

Figure 14: The decision tree of computing the complete answer to a CQ

7.3 Optimization of Conjunctive Queries

In this section we discuss how to optimize a CQ to compute its complete answer. We assume that the "most optimistic" planning strategy is used during the query planning process; that is, once there is some hope to compute the complete answer, the planning process continues traversing the decision tree. However, our discussions are also applicable to other planning strategies.

For a stable CQ Q (i.e., its minimal equivalent Q_m is feasible), an important optimization problem is how to find the cheapest feasible order of all the subgoals in Q_m under certain cost model. This problem of ordering subgoals can be viewed as the well known join-order problem (e.g., [AHY83, OL90, SG88]). A key difference between this subgoal-order problem and the traditional join-order problem is that in the former case, we do not need to consider all the orders of the subgoals (the number of which can be exponential). Instead, we consider only all the feasible orders, the space of which tends to be smaller. In other words, the relation binding patterns help us trim down the size of plan-search space. Furthermore, many fast-join algorithms (e.g., hash join, sort join) are more difficult to implement in the presence of binding patterns, and thus query optimization is trickier.

[FLMS99, YLUGM99] studied the subgoal-order problem. However, neither study considered the minimization of a query before checking its feasibility. If a query does not have a feasible order (e.g., the query Q_2 in Example 1.2), neither would answer the query, and they may miss the chance of computing the complete answer to a query. However, after minimizing a query, the techniques of both studies become applicable.

For a nonstable CQ Q, we can answer its minimal equivalent Q_m in two steps. (1) Use a linear plan to solve all the answerable subgoals Φ_a of Q_m, and compute the supplementary relation I_a. (2) Use an exhaustive plan to retrieve tuples for the nonanswerable subgoals Φ_{na} of Q_m. The first step can be treated as answering a stable query, i.e., the answerable subquery of Q_m. Thus the optimization techniques for stable queries can be used to optimize this subquery.

The execution of the exhaustive plan in step (2) can be recursive (as shown by [LC00, LC01a]), since we may access the relations repeatedly to retrieve more bindings, and with these bindings we can retrieve more tuples, and then more bindings, and so on. Since there can be many relations in the database, an important optimization problem for the second step is to decide what relations need to be accessed to provide useful bindings. As there can be many relations with different schemas and different binding patterns, it is important to include judiciously only those relations that can really contribute to the results of a query. [LC00, LC01a] studied how to find all the useful relations for a CQ to compute its maximal answer.

8 Conclusion

In this paper we studied a fundamental problem of answering queries in the presence of limited access patterns to relations: can the complete answer to a query be computed given the binding restrictions? If so, how to compute it? We studied this problem for various classes of queries, including conjunctive queries, unions of conjunctive queries, and conjunctive queries with arithmetic comparisons. We gave algorithms and complexity results for these classes. For datalog programs, we showed the problem is undecidable, and gave a sufficient condition for the computability of the complete answer to a query. Finally, we studied data-dependent computability of the complete answer to a query, and proposed a decision tree for guiding the process to compute the complete answer to a conjunctive query.

8.1 Discussions

There are several other classes of queries whose stability needs more investigation. One class is unions of conjunctive queries with arithmetic comparisons. For example, consider the query that is the union of the following two queries.

$$\begin{array}{lll} Q_1\text{:} & ans(X) & \text{:- } p(1, X), p(Y, Z), Y \leq Z. \\ Q_2\text{:} & ans(X) & \text{:- } p(1, X), p(Y, Z), Y \geq Z. \end{array}$$

Clearly this query is equivalent to the query:

$$Q' : ans(X) \text{ :- } p(1, X).$$

Suppose the only binding pattern for relation p is *bf*. Both Q_1 and Q_2 are not stable. However, the equivalent conjunctive query Q' of $Q_1 \cup Q_2$ is stable. This example shows that while considering the stability of a union of conjunctive queries with arithmetic comparisons, we cannot just test the stability of each query in the union. Instead, we should consider the possibility that some subset of the queries can form a query that is stable.

Another class of queries is unions of conjunctive queries with negations. For instance, consider query

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

Suppose both relations r and p have only one binding pattern *bf*. We can show that Q_1 is not stable. Intuitively, we cannot tell if there is a tuple $\langle y, z \rangle$ in relation r, and this tuple is not in relation p. On the other hand, the following query

$$Q_2 : ans(X) \text{ :- } r(a, X), r(Y, Z), p(Y, Z)$$

is not stable either. However, the union of the two queries is equivalent to the query

$$ans(X) \text{ :- } r(a, X)$$

which is stable. This example shows that, similar to the case of unions of conjunctive queries with arithmetic comparisons, we cannot check the stability of a union query by simply testing the stability of each query in the union. Instead, we should consider the possibility that a subset of the queries could be stable. The stability of these query classes needs future investigation.

Acknowledgments

The author thanks the anonymous reviewers for their valuable comments on an earlier version of this article.

References

[AHY83] Peter M. G. Apers, Alan R. Hevner, and S. Bing Yao. Optimization algorithms for distributed queries. *IEEE Transactions on Software Engineering (TSE)*, 9(1):57–68, 1983.

[ALM02] Foto Afrati, Chen Li, and Prasenjit Mitra. Answering queries using views with arithmetic comparisons. *PODS*, 2002.

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

[ASU79b] Alfred V. Aho, Yehoshua Sagiv, and Jeffrey D. Ullman. Equivalences among relational expressions. *SIAM Journal on Computing*, 8(2):218–246, 1979.

[BR87] Catriel Beeri and Raghu Ramakrishnan. On the power of magic. In *PODS*, pages 269–283, 1987.

[C+94] Sudarshan S. Chawathe et al. The TSIMMIS project: Integration of heterogeneous information sources. *IPSJ*, pages 7–18, 1994.

[CGKV88] Stavros S. Cosmadakis, Haim Gaifman, Paris C. Kanellakis, and Moshe Y. Vardi. Decidable optimization problems for database logic programs. *STOC*, pages 477–490, 1988.

[Cin] Cinemachine. `http://www.cinemachine.com/`.

[CM77] Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. *STOC*, pages 77–90, 1977.

[DL97] Oliver M. Duschka and Alon Levy. Recursive plans for information gathering. In *IJCAI*, 1997.

[Dus97] Oliver M. Duschka. Query planning and optimization in information integration. *Ph.D. Thesis, Computer Science Dept., Stanford Univ.*, 1997.

[FKL97] Daniela Florescu, Daphne Koller, and Alon Y. Levy. Using probabilistic information in data integration. In *Proc. of VLDB*, pages 216–225, 1997.

[FLMS99] Daniela Florescu, Alon Levy, Ioana Manolescu, and Dan Suciu. Query optimization in the presence of limited access patterns. In *SIGMOD*, pages 311–322, 1999.

[GGH98] Erich Grädel, Yuri Gurevich, and Colin Hirch. The complexity of query reliability. In *PODS*, pages 227–234, 1998.

[GMSV93] Haim Gaifman, Harry G. Mairson, Yehoshua Sagiv, and Moshe Y. Vardi. Undecidable optimization problems for database logic programs. *Journal of the ACM*, pages 683–713, 1993.

[GSUW94] Ashish Gupta, Yehoshua Sagiv, Jeffrey D. Ullman, and Jennifer Widom. Constraint checking with partial information. In *PODS*, pages 45–55, 1994.

[HKWY97] Laura M. Haas, Donald Kossmann, Edward L. Wimmers, and Jun Yang. Optimizing queries across diverse data sources. In *Proc. of VLDB*, pages 276–285, 1997.

[IFF+99] Zachary Ives, Daniela Florescu, Marc Friedman, Alon Levy, and Dan Weld. An adaptive query execution engine for data integration. In *SIGMOD*, pages 299–310, 1999.

[IMD] IMDB. The Internet Movie Database Ltd. Search Engine, `http://www.imdb.com/search/`.

[Ioa85] Yannis E. Ioannidis. A time bound on the materialization of some recursively defined views. In Alain Pirotte and Yannis Vassiliou, editors, *Proc. of VLDB*, pages 219–226. Morgan Kaufmann, 1985.

[JK83] David S. Johnson and Anthony C. Klug. Optimizing conjunctive queries that contain untyped variables. *SIAM Journal on Computing*, 12(4):616–640, 1983.

[KL88] Michael Kifer and Ai Li. On the semantics of rule-based expert systems with uncertainty. In Marc Gyssens, Jan Paredaens, and Dirk Van Gucht, editors, *ICDT*, volume 326 of *Lecture Notes in Computer Science*, pages 102–117, 1988.

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

[LC00] Chen Li and Edward Chang. Query planning with limited source capabilities. In *ICDE*, pages 401–412, 2000.

[LC01a] Chen Li and Edward Chang. Answering queries with useful bindings. *ACM Transactions on Database Systems (TODS)*, 26(3):313–343, 2001.

[LC01b] Chen Li and Edward Chang. On answering queries in the presence of limited access patterns. In *ICDT*, pages 99–113, 2001.

[Lev96] Alon Levy. Obtaining complete answers from incomplete databases. In *Proc. of VLDB*, pages 402–412, 1996.

[Li99] Chen Li. Computing complete answers to queries in the presence of limited access patterns (extended version). *Technical report, Computer Science Dept., Stanford Univ.,* `http://dbpubs.stanford.edu:8090/pub/1999-11`, 1999.

[LMSS95] Alon Levy, Alberto O. Mendelzon, Yehoshua Sagiv, and Divesh Srivastava. Answering queries using views. In *PODS*, pages 95–104, 1995.

[LRO96] Alon Levy, Anand Rajaraman, and Joann J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proc. of VLDB*, pages 251–262, 1996.

[LYV+98] Chen Li, Ramana Yerneni, Vasilis Vassalos, Hector Garcia-Molina, Yannis Papakonstantinou, Jeffrey D. Ullman, and Murty Valiveti. Capability based mediation in TSIMMIS. In *SIGMOD*, pages 564–566, 1998.

[MAM+98] Giansalvatore Mecca, Paolo Atzeni, Alessandro Masci, Paolo Merialdo, and Giuseppe Sindoni. The Araneus web-base management system. In *SIGMOD*, pages 544–546, 1998.

[MM01] Alberto O. Mendelzon and George A. Mihaila. Querying partially sound and complete data sources. In *PODS*, 2001.

[Mor88] Katherine A. Morris. An algorithm for ordering subgoals in NAIL! In *PODS*, pages 82–88, 1988.

[NS87] Jeffrey F. Naughton and Yehoshua Sagiv. A decidable class of bounded recursions. In *PODS*, pages 227–236. ACM, 1987.

[OL90] Kiyoshi Ono and Guy M. Lohman. Measuring the complexity of join enumeration in query optimization. In *Proc. of VLDB*, pages 314–325. Morgan Kaufmann, 1990.

[Qia96] Xiaolei Qian. Query folding. In *ICDE*, pages 48–55, 1996.

[RSU95] Anand Rajaraman, Yehoshua Sagiv, and Jeffrey D. Ullman. Answering queries using templates with binding patterns. In *PODS*, pages 105–112, 1995.

[Sag85] Yehoshua Sagiv. On computing restricted projections of representative instances. In *PODS*, pages 171–180. ACM, 1985.

[Sar91] Yatin Saraiya. Subtree elimination algorithms in deductive databases. *Ph.D. Thesis, Computer Science Dept., Stanford Univ.*, 1991.

[SG88] Arun N. Swami and Anoop Gupta. Optimization of large join queries. In *SIGMOD*, pages 8–17, 1988.

[Shm93] Oded Shmueli. Equivalence of datalog queries is undecidable. *Journal of Logic Programming*, 15(3):231–241, 1993.

[SY80] Yehoshua Sagiv and Mihalis Yannakakis. Equivalences among relational expressions with the union and difference operators. *Journal of the ACM*, 27(4):633–655, 1980.

[Ull89] Jeffrey D. Ullman. *Principles of Database and Knowledge-base Systems, Volumes II: The New Technologies*. Computer Science Press, New York, 1989.

[UV88] Jeffrey D. Ullman and Moshe Y. Vardi. The complexity of ordering subgoals. In *PODS*, pages 74–81, 1988.

[YLGMU99] Ramana Yerneni, Chen Li, Hector Garcia-Molina, and Jeffrey D. Ullman. Computing capabilities of mediators. In *SIGMOD*, pages 443–454, 1999.

[YLUGM99] Ramana Yerneni, Chen Li, Jeffrey D. Ullman, and Hector Garcia-Molina. Optimizing large join queries in mediation systems. In *ICDT*, pages 348–364, 1999.

[YÖL97] Ling-Ling Yan, M. Tamer Özsu, and Ling Liu. Accessing heterogeneous data through homogenization and integration mediators. In *CoopIS*, pages 130–139, 1997.

[ZO93] Xubo Zhang and Meral Ozsoyoglu. On efficient reasoning with implication constraints. In *DOOD*, pages 236–252, 1993.