# Computing Complete Answers to Queries in the Presence of Limited Access Patterns (Extended Version)

Chen Li
Department of Computer Science, Stanford University, CA 94305
chenli@db.stanford.edu

**Abstract**

We consider the problem of computing the complete answer to a query when there is limited access to relations, i.e., when binding patterns require values to be specified for certain attributes in order to retrieve data from a relation. This problem is common in information-integration systems, where heterogeneous sources have diverse and limited query capabilities. A query is *stable* if for any instance of the relations mentioned in the query, the complete answer to the query can be computed, using only the binding patterns permitted for queries at the various sources. We first study conjunctive queries, and we show that a conjunctive query is stable if and only if its minimal equivalent query $Q_m$ has an order of all the subgoals in $Q_m$, such that each subgoal in the order can be queried with a legal binding pattern. We propose two algorithms for testing stability of conjunctive queries, and we prove this problem is $\mathcal{NP}$-complete. For a nonstable conjunctive query, whether its complete answer can be computed is data dependent. We develop a decision tree that guides the query planning process to compute the complete answer to a conjunctive query, if it can be computed at all. We also study stability of unions of conjunctive queries, and conjunctive queries with arithmetic comparisons. In both cases we propose algorithms for testing stability of queries. Finally, we study datalog queries, and prove that if a set of rules and a query goal have a feasible rule/goal graph, then the query is stable. We also prove that stability of datalog programs is undecidable.

**Keywords**: complete answers to queries, binding restrictions, information-integration systems, query containment, query equivalence, datalog programs.

## 1 Introduction

The goal of information-integration systems is to support seamless access to heterogeneous data sources. Many systems (e.g., [C+94, HKWY97, IFF+99, LRO96, MAM+98]) have been proposed to reach this goal. In heterogeneous environments, especially in the context of World Wide Web, sources have diverse and limited query capabilities. For instance, many Web movie sources like The IMDB [IMD] and Cinemachine [Cin] provide search forms for movie information. A user fills out a form 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. Due to the source restrictions, we cannot retrieve all possible information from sources.

In this paper 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 all relations. 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** Assume that we have two relations. Relation $r(Star, Movie)$ has information about movies and their stars. Its only binding pattern, $bf$, says that each query to $s_1$ must specify a star name.[1] Similarly, relation $s(Movie, Award)$ has a binding pattern $bf$. Consider the following query that asks for the awards of the movies in which Henry Fonda starred:

$Q_1$: `ans(A) :- r('Henry Fonda',M) & s(M,A)`

To answer $Q_1$, we first access relation $r$ to retrieve the movies in which Henry Fonda starred. For each returned movie, access relation $s$ to obtain its awards. Return all these awards as the answer to the query. Note that the first subgoal must be solved before the second due to the relation restrictions. Although only part of the two relations was retrieved, we can still claim that the computed answer is the complete answer. The reason is that all the tuples of relation $r$ that satisfy the first subgoal were retrieved in the first step. Similarly, 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. □

Query $Q_1$ is called a "stable" query. A query is *stable* if for any instance of the relations mentioned in the query, the complete answer to the query is computable. That is, there exists a plan such that the answer computed by this plan is guaranteed to be the complete answer to the query. (The formal definition is in Section 2.) We show that if a conjunctive query is *feasible* (i.e., it has an order of all its subgoals such that the variables bound by the previous subgoals provide enough bound arguments that the relation for the subgoal can be accessed using a legal binding pattern), then the query is stable, and its complete answer can be computed by a linear plan (details in Section 3). In addition, the following example shows that a query can be stable even if it is not feasible.

**EXAMPLE 1.2** Suppose that we have two relations: $r(A, B, C)$ with a binding pattern $bff$, and $s(D, E, F)$ with a binding pattern $fbb$. Consider the following query:

$Q_2$: `ans(V,X) :- r(a,V,Y) & r(b,W,U) & s(X,V,W) & s(X,Z,W)`

This query is not feasible, since the binding pattern $fbb$ of relation $s$ requires the second argument to be bound, but variable $Z$ in subgoal $s(X, Z, W)$ cannot be bound by other subgoals. However, this subgoal is actually redundant, and $Q_2$ is equivalent to the following query:[2]

$Q_2'$: `ans(V,X) :- r(a,V,Y) & r(b,W,U) & s(X,V,W)`

$Q_2'$ is contained in $Q_2$ (denoted $Q_2' \subseteq Q_2$) because there is a containment mapping [CM77] from $Q_2$ to $Q_2'$: $a \rightarrow a$, $b \rightarrow b$, $V \rightarrow V$, $Y \rightarrow Y$, $W \rightarrow W$, $U \rightarrow U$, $X \rightarrow X$, and $Z \rightarrow V$. Similarly, $Q_2 \subseteq Q_2'$ because the identity mapping on subgoals gives us a containment mapping from $Q_2'$ to $Q_2$. Since $Q_2'$ has a feasible order of all its subgoals: $r(a, V, Y)$, $r(b, W, U)$, and $s(X, V, W)$, a linear plan following this order can compute the complete answer, which is also the complete answer to $Q_2$. □

[1] Binding restrictions of relations are described as binding patterns with $bf$ attribute adornments: $b$ (the attribute must be bound) and $f$ (the attribute can be free).

[2] Two queries are equivalent if they produce the same answer for any database.

Example 1.2 suggests that we need to minimize a conjunctive query before checking its feasibility. However, even if the minimal equivalent of a query $Q$ is not feasible, it is still not clear whether the query has an equivalent query that is feasible. Note that there is no a priori limit on the size of an equivalent of query $Q$, and some equivalents may look quite different from $Q$. Fortunately, in Section 3 we prove that if a minimal conjunctive query is not feasible, then no equivalent of the query can be feasible. We then show that a conjunctive query is stable if and only if its minimal equivalent is feasible. In particular, if its minimal equivalent is not feasible, then there can always be a database, such that the complete answer to the query cannot be computed. We propose two algorithms for testing stability of conjunctive queries, and we prove this problem is $\mathcal{NP}$-complete.

For a nonstable conjunctive query, whether its complete answer can be computed is data dependent. We categorize nonstable conjunctive queries into two classes based on whether the distinguished variables can be bound by the answerable subgoals in a query. We thus develop a decision tree (as shown in Figure 8) that guides the planning process to compute the complete answer to a conjunctive query. While traversing the decision tree, two planning strategies — a pessimistic strategy and an optimistic strategy — can be taken. What strategy should be taken depends on the application.

We then study unions of conjunctive queries, and show similar results as conjunctive queries (Section 5). In particular, we need to minimize a union of conjunctive queries before checking its stability. We show that a finite union of conjunctive queries $\mathcal{Q}$ is stable iff each query in the minimal equivalent of $\mathcal{Q}$ is stable. We also propose two algorithm for testing stability of unions of conjunctive queries. For conjunctive queries with arithmetic comparisons, stability testing becomes tricky since a conjunctive query might not have a minimal equivalent consisting of a subset of its subgoals. We give an algorithm for testing stability of conjunctive queries with arithmetic comparisons. Finally we study datalog queries [Ull89], and show that if a set of rules and a query goal have a feasible rule/goal graph [Ull89], then the query is stable.

Here is a summary of the contributions of this study:

1. We show that a conjunctive query (CQ for short) is stable iff its minimal equivalent is feasible. We propose two algorithms for testing stability of CQ's, and we prove this problem is $\mathcal{NP}$-complete.
2. For a nonstable CQ, whether its complete answer can be computed is data dependent. We propose a decision tree to guide the planning process to compute the complete answer to a CQ.
3. We study stability of unions of conjunctive queries (UCQ's for short), and give similar results as CQ's. We propose two algorithms for testing stability of UCQ's,
4. Testing stability of CQ's with arithmetic comparisons (CQAC's for short) is more challenging because equalities in a CQAC may help bind more variables, and then make more subgoals answerable. In addition, a CQAC may not have a minimal equivalent that has a subset of its subgoals. We propose an algorithm for testing stability of CQAC's.
5. For datalog queries, we show that if a set of rules has a feasible rule/goal graph with respect to a query goal, then the query is stable. We also prove that stability of datalog programs is undecidable.

### 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.

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 four 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] 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 4, we can sometimes use the approach in that paper to compute the complete answer to a nonstable conjunctive query.

The dynamic case of computing a complete answer to a nonstable query 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. As we will see in Section 4, in our framework we use bound-free adornments to describe relation restrictions. 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. That is why the computability of the complete answer is data dependent.

# 2 Preliminaries

In this section, we introduce the notation used throughout the paper. We also discuss the plan space considered in our study, especially linear plans and exhaustive plans.

## 2.1 Queries and Binding Restrictions

We first consider the class of conjunctive queries, then we will study unions of conjunctive queries, conjunctive queries with arithmetic comparisons, and datalog queries. A conjunctive query (CQ for short), also known as a select-project-join query, is denoted by:

$$h(\overline{X}) \text{ :- } g_1(\overline{X_1}) \ \& \ \ldots \ \& \ g_n(\overline{X_n})$$

The query's body includes the *subgoals* $g_1(\overline{X_1}), \ldots, g_n(\overline{X_n})$. Each subgoal $g_i(\overline{X_i})$ includes a relation $g_i$ (also called a source), and a tuple of arguments $\overline{X_i}$ corresponding to the relation schema. Each argument in a subgoal can be either a variable or a constant. The predicate $h(\overline{X})$ in the head represents the results of the query. The variables $\overline{X}$ are called *distinguished* variables. We consider *safe* CQ's, i.e., every distinguished variable appears in the body. We use names beginning with lower-case letters for constants and relation names, and names beginning with upper-case letters for variables.

Each relation $R_i$ is associated with a set of $bf$ binding patterns $\mathcal{T}_i = \{p_{i1}, \ldots, p_{ik_i}\}$. Each binding pattern $p_{ij}$ represents a form of the possible queries that are acceptable by the relation. For instance, a relation $r(A, B, C)$ with the binding patterns $\mathcal{T} = \{bff, ffb\}$ requires that every query to the relation must either supply a value for the first argument, or supply a value for the third argument. Note that if an argument is adorned $f$ in a binding pattern, it may still be bound to a value in a query. Consequently some binding patterns are more general than others, in the sense that every query that is acceptable by one pattern is also acceptable by another. For example, binding pattern $bff$ is more general than $bbf$. The following observation serves as a starting point of our work.

**Observation 1** *If a relation does not have an all-free binding pattern, then after some 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.* □

Figure 1 shows the main idea of the observation. The shaded tuples in the figure are the tuples that have been retrieved by some source queries. (We describe our assumptions of obtaining bindings shortly.)

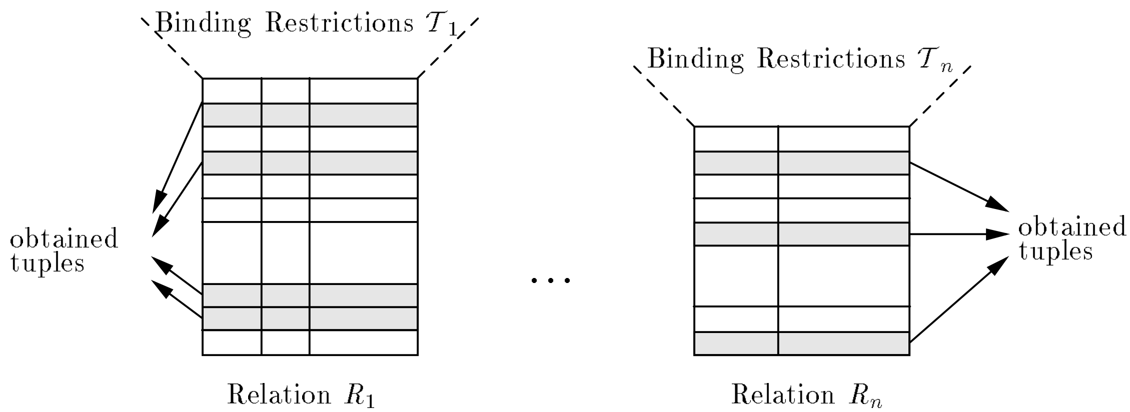


Figure 1: Obtained tuples in a database

## 2.2 Stable Queries and Plans

**Definition 2.1** (**complete answer to a query**) Given a database $D$ of relations with binding restrictions, the *complete* answer to a query $Q$, denoted by $ANS(Q, D)$, is the query's answer that could be computed if we could retrieve all the tuples from the relations. □

**Definition 2.2** (**plan**) A *plan* for a query is an algorithm in which sources are only asked queries with legal binding patterns. □

Formally, if a plan includes a source query $R_i(a_1, \ldots, a_k, A_{k+1}, \ldots, A_n)$ to a relation $R_i$, in which $a_1, \ldots, a_k$ are constants, and $A_{k+1}, \ldots, A_n$ are variables, then the relation should have a binding pattern in which the arguments for $A_{k+1}, \ldots, A_n$ are all adorned $f$. A legal plan can preprocess the data from the relations in various ways, e.g., by doing join, projection, or selection.

**Definition 2.3** (**stable query**) A query on relations with binding restrictions is *stable* if there exists a plan for the query that computes the query's complete answer independent of the database. □

In this study we are especially interested in the following two classes of plans: linear plans and exhaustive plans.

### 2.2.1 Linear Plans

**Definition 2.4 (feasible order of subgoals)** Some subgoals $g_1(\overline{X_1}), \ldots, g_k(\overline{X_k})$ in a CQ form a *feasible order* if for each subgoal $g_i(\overline{X_i})$ in the order, given the variables that are bound by the previous subgoals, subgoal $g_i(\overline{X_i})$ is *answerable*; that is, there is a binding pattern $p_{ij}$ of the relation $g_i$, such that for each argument $X$ in subgoal $g_i(\overline{X_i})$ that is adorned as $b$ in $p_{ij}$, either $X$ is a constant, or $X$ appears in a previous subgoal. A CQ is *feasible* if it has a feasible order of *all* its subgoals. □

**EXAMPLE 2.1** Consider the queries in Example 1.2. Query $Q_2'$ is feasible, since it has a feasible order $r(a,V,Y)$, $r(b,W,U)$, and $s(X,V,W)$ of all its subgoals. The first argument of the first subgoal $r(a,V,Y)$ is a constant $a$, and relation $r$ has a binding pattern $bff$, so this subgoal is answerable by relation $r$. Similarly, the second subgoal is also answerable by relation $r$. The relation $s$ for the third subgoal $s(X,V,W)$ has a binding pattern $fbb$. Since the variables $V$ and $W$ in this subgoal appear in the first two subgoals, this subgoal is also answerable by relation $s$. Query $Q_2$ is not feasible, since it does not have a feasible order of all its subgoals. The example also shows that two equivalent queries can have different feasibility. □

Let a feasible CQ $Q$ have a feasible order of all its subgoals. A *linear plan* following this order corresponds to a left-deep tree of execution. Figure 2 shows linear plans for several queries: (a) for query $Q_1$ in Example 1.1, (b) for query $Q_2'$ in Example 1.2, and (c) for a general CQ with a feasible order of subgoals $g_{i_1}(\overline{X_{i_1}}), \ldots, g_{i_n}(\overline{X_{i_n}})$. We use the same notation of plans as in [FLMS99]: operator "$\bowtie$" stands for a regular join, and operator "$\overrightarrow{\bowtie}$" stands for a dependent join.

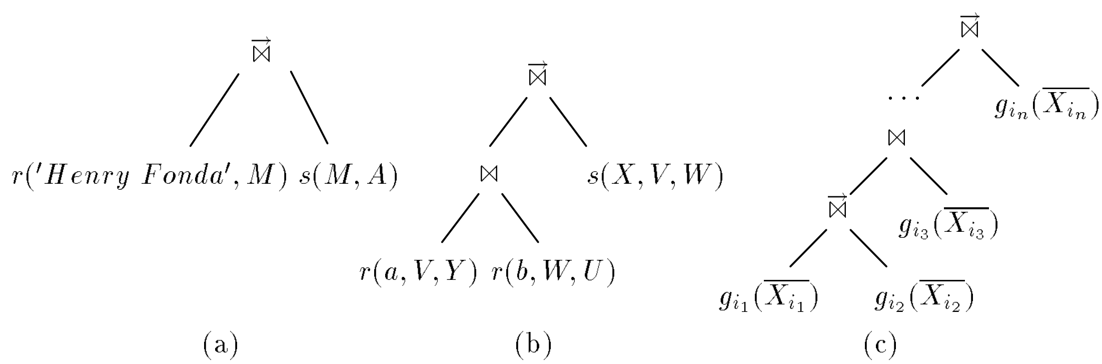


Figure 2: Linear Plans

If a feasible CQ $Q$ have a feasible order $g_1(\overline{X_1}), \ldots, g_n(\overline{X_n})$ of all its subgoals. A linear plan for the query following this order computes the corresponding sequence of $n$ *supplementary relations* [BR87] $I_1, \ldots, I_n$. The arguments of each supplementary relation $I_j$ correspond to those variables that are both bound and relevant after the first $j$ subgoals have been solved. A variable is *relevant* if it appears either in the head or in the $(j+1)$st or a subsequent subgoal. The value of relation $I_j$ is the set of bindings of these bound and relevant variables after the first $j$ subgoals have been answered. The answer to the query computed by the linear plan is the supplementary relation $I_n$.

**EXAMPLE 2.2** Consider the feasible order for the query $Q_2'$ in Example 1.2, and its corresponding linear plan. The schema of the supplementary relation $I_1$ is $V$, since variable $V$ is bound by the first subgoal. $V$ is relevant since it is used both in the third subgoal and in the head. Variable $Y$ is not included in the schema because, although it is bound by the first subgoal, it appears neither in other subgoals, nor in the head. Similarly, $VW$ and $VX$ are the schemas of the supplementary relations $I_2$ and $I_3$, respectively. The answer computed by this linear plan is the supplementary relation $I_3$. □

Each supplementary relation $I_j$ is computed as follows. Choose a binding pattern $p$ of relation $g_j$ whose binding requirements are satisfied by the constants in subgoal $g_j(\overline{X_j})$ and the variables in $I_{j-1}$. If the constant arguments of subgoal $g_j(\overline{X_j})$ alone can satisfy the binding pattern $p$ (which should be true for the first subgoal $g_1(\overline{X_1})$), then send a source query to relation $g_j$ by binding the constants to the corresponding arguments. Join the returned tuples with supplementary relation $I_{j-1}$ (if it exists), and project the result onto the relevant variables to compute $I_j$.

If the binding requirements of binding pattern $p$ can be satisfied by the constant arguments in subgoal $g_j$ plus a set of bound variables $\mathcal{B}$ in $I_{j-1}$, compute the relation $I_{j-1}^{\mathcal{B}}$ by projecting $I_{j-1}$ onto the variables $\mathcal{B}$. We bind each tuple $t$ in the relation $I_{j-1}^{\mathcal{B}}$, together with the constants in subgoal $g_j$, to the corresponding arguments in subgoal $g_j$, and send a source query to relation $g_j$. Join the returned tuples with $I_{j-1}$, and project the result onto the relevant variables.[3] Finally, take the union of the results for all the tuples in relation $I_{j-1}^{\mathcal{B}}$ to compute $I_j$.

Note that by choosing different binding patterns to compute $I_j$, we have different executions with different costs. Which binding pattern to choose should be considered in the cost-based optimization. In addition, we should also consider how to choose the cheapest feasible order of all the subgoals, since there can be multiple feasible orders. We will discuss these optimization issues in Section 4.4. Clearly a linear plan for a query exists if the query is feasible. As shown in Section 3, for any instance of the relations mentioned in a feasible CQ, the answer computed by a linear plan is the complete answer to the query.

### 2.2.2 Exhaustive Plans

An *exhaustive plan* for a query accesses all possible relations to obtain bindings, and retrieves as many tuples from the relations as possible to answer the query. The following is an example.

**EXAMPLE 2.3** Assume that we have three relations: relation $r(Star, Movie)$ with a binding pattern $fb$, relation $s(Movie, Award)$ with a binding pattern $bf$, and relation $T(Movie, Studio)$ with a binding pattern $ff$. Consider the following query $Q$, which is the same as the query $Q_1$ in Example 1.1. (Note that the binding pattern of $r$ is $fb$, not $bf$ as in Example 1.1.)

$Q$: `ans(A) :- r('Henry Fonda',M) & s(M,A)`

This query is not feasible, and it cannot be answered using only the two relations $r$ and $s$ that are mentioned in the query. However, we can query relation $T$, whose binding pattern is $ff$, to retrieve some movies. For each movie, we query relation $r$ to see whether Henry Fonda starred in it. If so, we query relation $s$ to retrieve its awards. Thus we can still compute some answers to the query by

[3] The join is called in the literature a dependent join, a functional join, implicit join, filter join, theta semi-join, bind join, and etc.

accessing relation $T$, although $T$ is not mentioned in the query. Note that the movies returned from $T$ do not necessarily include all the movies in relation $r$, thus we cannot guarantee whether the computed answer is the complete answer. $\square$

Example 2.3 shows that relations not mentioned in a query can be accessed to contribute bindings to compute the query's results. An *exhaustive plan* accesses all possible relations to obtain as many bindings as possible, and answer the query using the retrieved tuples. [LC00] showed how to implement an exhaustive plan by translating restrictions and a CQ into a datalog program, and evaluating the program on the relations. datalog is used in the planning process since the process can be recursive (i.e., we may access the relations repeatedly to obtain more bindings), although the query itself is not.

In general, an exhaustive plan for a CQ is more expensive than a linear plan, since an exhaustive plan often accesses relations not mentioned in the query to obtain bindings. Thus adding more relations to the database may help compute more results for the query. [LC00] also discussed how to incorporate cached data into an exhaustive plan by adding the corresponding rules. As we will see in Section 4, exhaustive plans are especially useful for nonstable CQ's, since these queries cannot be answered by linear plans. Since the complete answer to a stable query can be computed by a linear plan, an exhaustive plan is not necessary for stable CQ's.

### 2.3 Binding Assumptions

Throughout this study we make the following important binding assumptions:

1. Each binding for an attribute must be from the domain of this attribute.
2. If a relation requires a value, say, a string, as a particular argument, we will not allow the strategy of trying all the possible strings to "test" the relation.
3. Rather we assume that any binding for an attribute $A$ is either obtained from the user query, or from a tuple returned by another source query to a relation. In both cases the binding value comes from an attribute in the same domain as attribute $A$.

We use Example 2.3 to explain these assumptions. The first assumption says that we would not use a star name as a binding for attribute $Movie$. The binding pattern $bf$ of relation $s$ requires each query to relation $s$ should give a value in the domain of movie title. The second assumption says that we would not allow the following "approach": generate all possible strings to test whether relation $s$ has movies with these strings as titles. This approach would not terminate, since there will be an infinite number of strings that need to be tested. By the third assumption, if $m$ is a movie title returned from relation $T$, or is an initial binding for attribute $Movie$ in a user query, then $m$ can be used to query relation $s$.

## 3 Stability of Conjunctive Queries

In this section we study stability of CQ's. We propose two algorithms for testing stability of CQ's, and prove that this problem is $\mathcal{NP}$-complete.

### 3.1 Feasible Conjunctive Queries

The following lemma shows that all feasible CQ's are stable.

**Lemma 3.1** *If a CQ $Q$ on relations with binding restrictions is feasible, then for any database $D$, $ANS(Q, D)$ can be computed by a linear plan. Thus every feasible CQ is stable.* □

**Proof:** Let the feasible CQ $Q$ have a feasible order $g_1(\overline{X_1}), \ldots, g_n(\overline{X_n})$ of all its subgoals, and $L$ be the corresponding linear plan of this order. 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. This tuple agrees with the tuple $t_j$ on their common variables. Therefore, during the computation of the supplementary relation $I_j$ in the plan $L$, no matter which binding pattern of the relation $g_j$ is chosen, tuple $t_j$ in $g_j$ is retrieved by a source query 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 plan $L$ includes the tuple $t$, which can 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 4, whether the computability of the complete answer to a nonstable CQ is *dynamic*, since the computability is unknown until some plan is executed.

The feasibility of a CQ (i.e., the existence of a feasible order of its subgoals) can be checked by a greedy algorithm, called the *Inflationary* algorithm. That is, initialize the set $\Phi_a$ of answerable subgoals to be empty. With the variables bound by the subgoals in $\Phi_a$, whenever a subgoal becomes answerable by its relation (as defined in Section 2.2.1), add this subgoal to $\Phi_a$. Iterate by adding more subgoals to $\Phi_a$. The query is feasible if and only if all the subgoals in the query become answerable.

### 3.2 Minimal Equivalents of CQ's

Example 1.2 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 corollary.

**Corollary 3.1** *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 $\Theta_P = <e_1, \ldots, e_m>$ is a feasible order of all the subgoals in $P$. Since the two queries are equivalent, there exist two containment mappings $\mu$: $Q \rightarrow P$, and $\nu$: $P \rightarrow Q$. Consider the targets in $Q$ of the subgoals $e_1, \ldots, e_m$ under the mapping $\nu$: $\nu(e_1), \ldots, \nu(e_m)$. 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 $Q$: $\Theta_Q = <g_1, \ldots, g_n>$, as shown in Figure 3. Now we prove that $\Theta_Q$ is a feasible order of *all* the subgoals in $Q$. That is, we need to show: (1) $\Theta_Q$ includes all the subgoals in query $Q$; (2) $\Theta_Q$ is a feasible order. Since query $Q$ is not feasible, we can claim that the equivalent query $P$ actually does not exist.

$$Q:\quad H \text{ :- } g_1(\ldots) \;\&\; \ldots \;\&\; g_t(\ldots,X,\ldots) \;\&\; \ldots \;\&\; g_i(\ldots,X,\ldots) \;\&\; \ldots \;\&\; g_n(\ldots)$$

$$\mu \qquad \nu \qquad\qquad \uparrow \nu \qquad\qquad \uparrow \nu$$

$$P:\quad E \text{ :- } e_1(\ldots) \;\&\; \ldots \;\&\; e_j(\ldots,Y,\ldots) \;\&\; \ldots \;\&\; e_{k_i}(\ldots,Y,\ldots) \;\&\; \ldots \;\&\; e_m(\ldots)$$

Figure 3: Proof of Lemma 3.2

Claim (1) is correct because $Q$ is minimal. If $\Theta_Q$ did not include all the subgoals in $Q$, let $Q'$ be the head of $Q$ and the subgoals in $\Theta_Q$. Then $Q' \subseteq P$ because of the containment mapping $\nu$, and $P \subseteq Q'$ because of the containment mapping $\mu$. 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 $\Theta_Q$. Since the containment mapping $\nu$ 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 $e_1$.

Consider each subgoal $g_i$ in the order $\Theta_Q$, and let $g_i = \nu(e_{k_i})$ for some $1 \leq k_i \leq m$. Since subgoal $e_{k_i}$ is answerable in $\Theta_P$, 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 $\Theta_Q$ was constructed, there exists a subgoal $g_t$ before $g_i$ in $\Theta_Q$, such that $g_t = \nu(e_j)$. (If $\nu(e_j)$ was removed during the construction of $\Theta_Q$, then $g_t$ is a subgoal identical to $\nu(e_j)$.) 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 $\Theta_Q$. 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 query $Q$ has a minimal equivalent $Q_m$ that 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. 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. Since $Q_m$ is not feasible, $\Phi_{na}$ is not empty. Let this substitution turn the head of $Q_m$ to a tuple $t_h$.

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

Figure 4: Proof of Lemma 3.3

As shown in Figure 4, $D_1$ 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.[4] A relation may have multiple tuples, since it may appear in multiple subgoals of $Q_m$. All the relations that are not mentioned in the query are empty in both databases, so that these relations cannot provide any bindings.

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 $t_h \notin ANS(Q_m, D_1)$.[5] Otherwise, there must be a substitution $\tau$ from a subset of the obtainable tuples $\{t_1, \ldots, t_k\}$ to all the subgoals $g_1, \ldots, g_n$, such that under $\tau$ each subgoal becomes true. Let $Q'_m$ be a query with the head of $Q_m$ plus the subgoals of the tuples used in $\tau$. Since each variable was assigned with a distinct constant, these constants can represent their corresponding variables. Thus $\tau$ can be considered to be a containment mapping from $Q_m$ to $Q'_m$, and $Q_m = Q'_m$. 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 Algorithm: CQstable

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

Theorem 3.1 is correct because of Corollary 3.1 and Lemma 3.3. By this theorem, we give an algorithm *CQstable* for testing the stability of a CQ, as shown in Figure 5. Assume that a CQ $Q$ has $n$ subgoals, and its minimal equivalent $Q_m$ has $k$ subgoals. It is known that the minimization of CQ's 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

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

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

**Algorithm CQstable:** Test stability of CQ's
**Input:** • $Q$: A conjunctive query.
• $B$: Binding restrictions 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 5: Algorithm: CQstable

$O(k^2)$. Since $k \leq n$, the total time complexity of the algorithm CQstable is exponential in the size of $Q$.

### 3.4 Algorithm: CQstable*

The exponential complexity of the algorithm CQstable comes from the fact that we need to minimize a CQ before its feasibility. There is a more efficient algorithm for testing stability of CQ's, which is based on the following theorem:

**Theorem 3.2** *Let $Q$ be a CQ on relations with binding restrictions. Let $Q_a$ be its* answerable sub-query*, i.e., the query that includes the head of $Q$ and the answerable subgoals of $Q$. Then $Q$ is stable iff $Q = Q_a$.* □

**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 $Q'_m \neq Q_m$. ■

**Algorithm CQstable*:** Test stability of CQ's
**Input:** • $Q$: A conjunctive query.
• $B$: Binding restrictions 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 6: Algorithm: CQstable*

Theorem 3.2 gives another algorithm *CQstable** for testing stability of CQ's, as shown in Figure 6. One advantage of algorithm CQstable* 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 $O(n^2)$, where $n$ is the number of subgoals in $Q$. However, if not all the subgoal 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 6, is that we can extend it to CQ's with arithmetic comparisons (CQAC's for short). We cannot extend the algorithm CQstable to the case of CQAC's, since a CQAC does not necessarily have a unique minimal form (details in Section 6).

### 3.5 Complexity of Testing Stability of CQ's

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

**Theorem 3.3** *Stability of a CQ is $\mathcal{NP}$-complete.* □

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

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

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

(b) Testing the stability of $P$

Figure 7: 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 $Q' \subseteq Q$ is known to be $\mathcal{NP}$-complete [CM77]. We reduce this problem to the problem of testing the stability of a CQ. Let $Q$ be $ans(\overline{X})$ :- $g_1(\overline{X_1}) \& \ldots \& g_n(\overline{X_n})$, and $Q'$ be $ans(\overline{X})$ :- $g_1(\overline{X_1}) \& \ldots \& g_k(\overline{X_k})$, where $k < n$. We construct a query $P$ on relations with binding restrictions, such that $P$ is stable iff $Q' \subseteq Q$.

Figure 7 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 $h(A), g_1'(A, \overline{X_1}), \ldots, g_k'(A, \overline{X_k})$, $g_{k+1}'(B, \overline{X_{k+1}}), \ldots, g_n'(B, \overline{X_n})$.

Clearly the above construction of query $P$ takes time that is polynomial in the size of $Q$. By the construction of the relations $h$ and $g_1', \ldots, g_n'$, the answerable subgoals of $P$ are $h(A), g_1'(A, \overline{X_1}), \ldots, g_k'(A, \overline{X_k})$. Let $P'$ be the query with these answerable subgoals. By Theorem 3.2, $P$ is stable iff $P = P'$, i.e., there is a containment mapping $\nu$ from $P$ to $P'$. It is easy to show that $\nu$ exists iff there is a containment mapping from $Q$ to $Q'$, which is true iff $Q' \subseteq Q$. Note that any such containment mapping from $P$ to $P'$ must map both variables $A$ and $B$ to $A$. ■

## 4 Nonstable Conjunctive Queries

For a nonstable CQ, in some cases we may still compute its complete answer. However, the computability of its complete answer is data dependent. In this section we discuss in what cases the complete answer to a nonstable CQ 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 complete its complete answer.

## 4.1 Dynamic Cases

The following example shows that even if a CQ is not stable, its complete answer may still be computable, and we do not know the computability until some plan is executed.

**EXAMPLE 4.1** 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 $A, B, \ldots, E$ have different domains. Consider the following two queries:

$Q_1$: `ans(B) :- r(a,B,C) & s(C,D)`
$Q_2$: `ans(D) :- 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 source query returns three tuples: $<a, b_1, c_1>$, $<a, b_2, c_2>$, and $<a, b_2, c_3>$. The supplementary relation $I_1$ after the first subgoal has the schema $BC$, and contains three tuples: $<b_1, c_1>$, $<b_2, c_2>$, and $<b_2, c_3>$. 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 $<c_1, d_1>$ and $<c_2, d_2>$ 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 $I_1$. Tuples $<b_1, c_1>$, $<b_2, c_2>$, and $<b_2, c_3>$ are the only tuples in $I_1$, and their projection onto variable $B$ is $\{b_1, b_2\}$. Tuples $<b_1, c_1>$ and $<b_2, c_2>$ in $I_1$ can join with tuples $<c_1, d_1>$ and $<c_2, d_2>$ 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 $<c_2, d_2>$ from relation $s$, then we do not know whether $<b_2>$ is the complete answer or not, since we do not know whether relation $s$ has a tuple $<c_1, d>$ (for some $d$ value) that has not been retrieved. This tuple can join with the tuple $<b_1, c_1>$ 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 $I_1$ also includes three tuples $<b_1, c_1>$, $<b_2, c_2>$, and $<b_2, c_3>$. 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 $<c_1, d'>$ in relation $s$ that has not been retrieved, and $d'$ is in the complete answer to $Q_2$. 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 $\Phi_a$. If all the distinguished variables can be bound by $\Phi_a$, i.e., the answerable subquery of $Q_m$ is safe, we use a linear plan of a feasible order of $\Phi_a$ 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 $\Phi_{na}$.
   (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 $\Phi_{na}$ (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 $\Phi_a$, 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 4.1** *For a minimal CQ $Q_m$, if the supplementary relation $I_a$ of the answerable subgoals $\Phi_a$ 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 $\Phi_a$ that contribute to the tuple $t$. By using a linear plan of a feasible order of $\Phi_a$, 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 4.2** *Assume that $Q_m$ is a minimal CQ, and all the distinguished variables are bound by the answerable subgoals $\Phi_a$. 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 $t_1, \ldots, t_k$ of the answerable subgoals $\Phi_a$ and tuples $t_{k+1}, \ldots, t_n$ of the nonanswerable subgoals $\Phi_{na}$. Tuples $t_1, \ldots, t_k$ must be retrieved by a linear plan of a feasible order of $\Phi_a$ 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 $\Phi_a$. 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 $\Phi_{na}$ 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 $\Phi_{na}$ cannot be retrieved because of the restrictions of $\Phi_{na}$. Without these tuples, $t^P$ is not in the complete answer. Since we do not whether these tuples for $\Phi_{na}$ exist or not, we do not know whether the complete answer includes $t^P$ or not. ■

**Lemma 4.3** *For a minimal CQ $Q_m$, if not all the distinguished variables are bound by the answerable subgoals $\Phi_a$, 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 $\Phi_a$. Following the idea of the proof of Lemma 3.3, there can always be some tuples for the nonanswerable subgoals $\Phi_{na}$ 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 $\Phi_{na}$. Without these tuples, tuple $r$ is not in the complete answer. Since we do not whether these tuples for $\Phi_{na}$ 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 is dynamic or data dependent, since it is not known until some plan is executed, and some information about the relations becomes available.

### 4.2 The Decision Tree

We develop a decision tree (as shown in Figure 8) that guides the planning process to compute the complete answer to a CQ. The shaded nodes are where we can conclude about 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 8). Then we test the feasibility of the query $Q_m$ by calling the algorithm Inflationary; that is, we test whether $Q_m$ has a feasible order of all its subgoals. If so (arc 2 in Figure 8), $Q_m$ (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 $\Phi_a$ by calling the algorithm Inflationary. Then we check if all the distinguished variables are bound by the subgoals $\Phi_a$. There are two cases:

1. If all the distinguished variables are bound by the subgoals $\Phi_a$ (arc 4), then the complete answer to the query may be computed even if the supplementary relation $I_a$ of subgoals $\Phi_a$ is not empty. We compute $I_a$ by a linear plan following a feasible order of $\Phi_a$.
   (a) If $I_a$ is empty (arc 5), then the complete answer is empty.
   (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 $\Phi_{na}$, 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.
2. If some distinguished variables are not bound by the subgoals $\Phi_a$ (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.

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 the upper bound of the 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 $\phi$, 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.

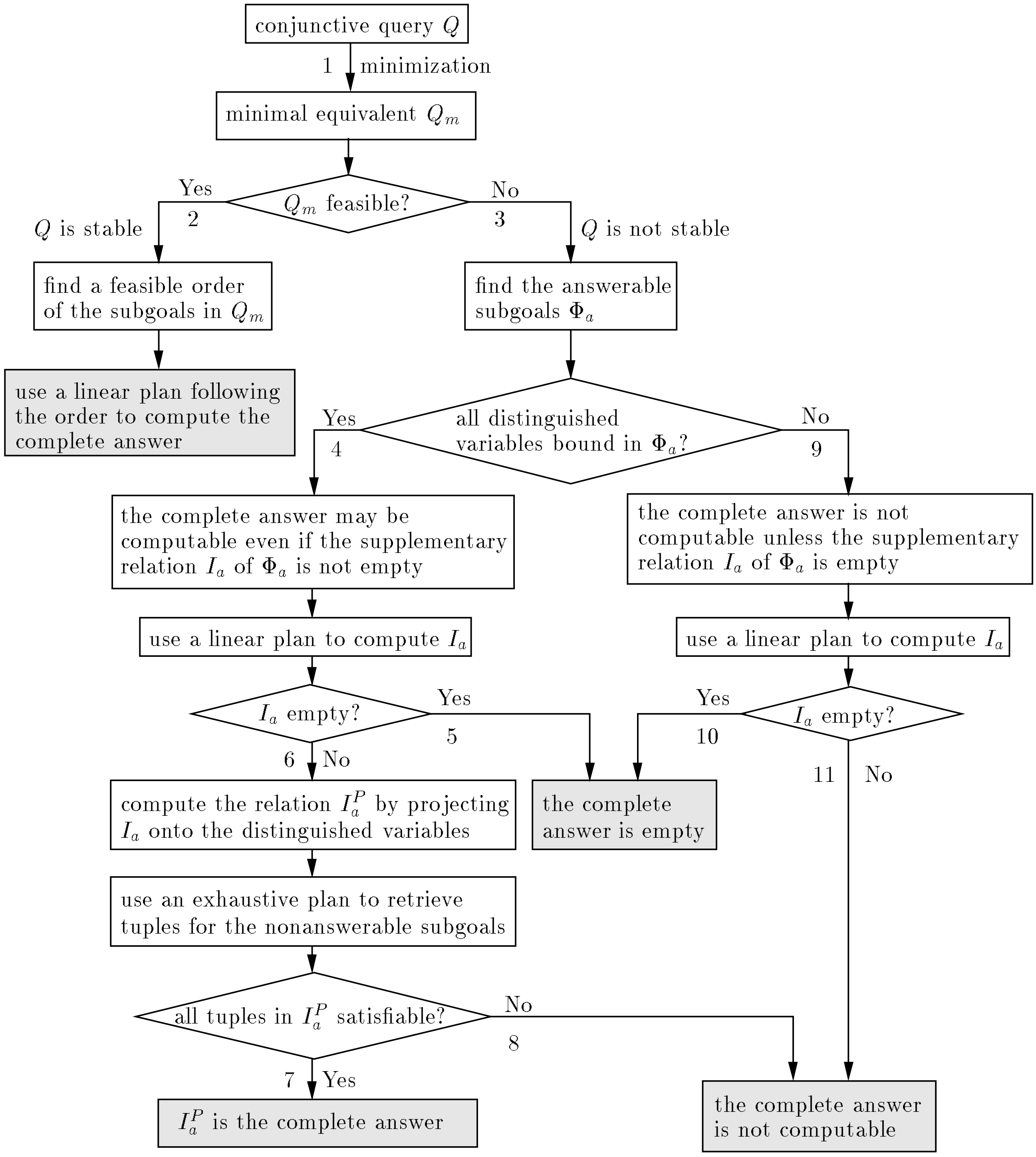


Figure 8: The decision tree of computing the complete answer to a CQ

## 4.3 Pessimistic Planning and Optimistic Planning

While traversing the decision tree from the root to a leaf node, we may reach a node where we do not know whether the complete answer is computable until we traverse one level down the tree. Two planning strategies can be adopted at this kind of nodes: a pessimistic strategy and an optimistic strategy. A *pessimistic* strategy gives up traversing the tree once the complete answer is unlikely to be computable. On the contrary, an *optimistic* strategy is optimistic about the possibility of computing the complete answer, and it traverses one more level by doing the corresponding operations.

For instance, if the minimal equivalent $Q_m$ is not feasible, and all the distinguished variables are bound by the answerable subgoals $\Phi_a$, we do not know whether the complete answer is computable before using a linear plan to compute the supplementary relation $I_a$ and checking the emptiness of $I_a$. A pessimistic strategy gives up the planning process, but claims that the complete answer cannot be computed. An optimistic strategy continues traversing the tree by executing a linear plan to compute $I_a$. If $I_a$ is empty, the complete answer is empty. Otherwise, we still have two options: a pessimistic strategy gives up answering the query, while an optimistic strategy executes an exhaustive plan to retrieve tuples for the nonanswerable subgoals $\Phi_{na}$.

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 it is that all the tuples in $I_a^P$ are satisfiable. We may use statistics to answer these questions and then make the decision about what strategy to take.

## 4.4 Optimization of CQ's

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 some 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 our ordering-subgoals problem and the traditional join-order problem is that 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.

[YLUGM99, FLMS99] studied the ordering-subgoals 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. Thus 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 $\Phi_a$ of $Q_m$, and compute the supplementary relation $I_a$; (2) Use an exhaustive plan to retrieve tuples for the nonanswerable subgoals $\Phi_{na}$ 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 the second step can be recursive (as shown by [LC00]), 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 shown by the following example, not all the relations that provide bindings can contribute to the query's results.

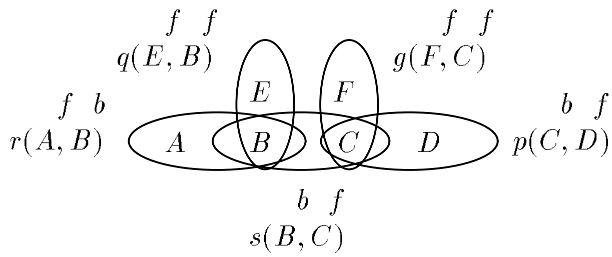


Figure 9: Useful relations for a query

**EXAMPLE 4.2** Figure 9 shows five relations in a database whose schemas share six attributes: $A, B, \ldots, F$. Their schemas are represented as a hypergraph [Ull89], in which each node is an attribute and each hyperedge is the schema of a source relation. Assume different attributes have different domains. Consider the following query:

$Q$: `ans(D) :- r(a,B) & s(B,C) & p(C,D)`

In words, the user knows the value of attribute $A$ is $a$, and wants to retrieve the associated $D$ values by $r \bowtie s \bowtie p$. Query $Q$ is not stable, since its minimal equivalent (itself) is not feasible. Thus the relations $q$ and $g$ that are not mentioned in the query may be useful, since they provide bindings to retrieve tuples from relations $r$, $s$, and $p$.

Although relation $g$ provides bindings for attribute $C$, it does *not* contribute to the results for the query. To illustrate the reason, we prove that all the results for the query that can be computed by using the five relations can be retrieved without using relation $g$. Suppose tuple $t = <d>$ is a tuple in the answer to $Q$ that is retrieved by using the five relations, and $p$ comes from tuple $t_1 = <a, b>$ of $r$, tuple $t_2 = <b, c>$ of $s$, and tuple $t_3 = <c, d>$ of $p$. Since only relation $q$ (with the binding pattern $ff$) takes attribute $B$ as a free attribute, the value $b$ of $B$ in tuple $t_1$ must be derived from a source query to $q$, which includes a tuple whose $B$ value is $b$. With $B = b$, we can retrieve tuple $t_1$ from $r$ by sending a source query $r(A, b)$, and retrieve tuple $t_2$ from $s$ by sending a source query $s(b, C)$. With the new binding $C = c$ in tuple $t_2$, we can retrieve tuple $t_3$ from $p$ by sending a source query $p(c, D)$. Therefore, without using relation $g$, we can compute the tuple $p$ in the answer to $Q$. The proof also shows that without using relation $q$, we cannot get any answer to the query. □

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] proposed an algorithm for finding all the useful relations for a CQ to compute its maximal obtainable answer.

# 5 Stability of Unions of Conjunctive Queries

In this section we study stability of unions of CQ's, and show similar results as CQ's. In particular, a union of CQ's is stable iff each query in its minimal equivalent is stable. We also propose two algorithms for testing of stability of unions of CQ's. 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 5.1** *Let $\mathcal{Q} = Q_1 \cup \cdots \cup Q_m$ and $\mathcal{R} = R_1 \cup \cdots \cup R_n$ be two UCQ's. Then $\mathcal{Q} \subseteq \mathcal{R}$ (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 $Q_i \subseteq R_j$.* □

**EXAMPLE 5.1** Let us see some examples of UCQ's 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:

$Q_1$: `ans(X) :- r(a,X)`
$Q_2$: `ans(X) :- r(a,X) & p(Y,Z)`
$Q_3$: `ans(X) :- r(a,X) & s(X,Y) & p(Y,Z)`

Clearly $Q_3 \subseteq Q_2 \subseteq Q_1$. Queries $Q_1$ and $Q_3$ are both stable (since they are both feasible), while query $Q_2$ is not. Consider the following two UCQ's: $\mathcal{Q}_1 = Q_1 \cup Q_2 \cup Q_3$ 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. □

## 5.1 Algorithm: UCQstable

In analogy with Theorem 3.1, we can prove a theorem that gives us a stability test for UCQ's.

**Theorem 5.2** *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 that the minimal equivalent $\mathcal{Q}_m$ has $k$ queries, $Q_1, \ldots, Q_k$, where $k \leq n$.

*If*: 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.

*Only If*: If $\mathcal{Q}_m$ has a query, say $Q_1$, that is not stable. Without loss of generality, suppose that subgoals $g_1(\overline{X_1}), \ldots, g_p(\overline{X_p})$ are the answerable subgoals of query $Q_1$, and subgoals $g_{p+1}(\overline{X_{p+1}}), \ldots, g_q(\overline{X_{q+1}})$ 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 $Q_1' \neq Q_1$. Consider the canonical tuples $t_1, \ldots, t_k$ of query $Q_1$ (see Section 3 for the definition of canonical tuples). Let $Q_1'$ be the query with the head of $Q_1$ and the answerable subgoals of $Q_1$. Let these tuples 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 need to prove 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 $\mathcal{Q}_m$ includes tuple $t_h$. By Theorem 3.2, $t_h$ cannot come from query $Q_1$, since otherwise there will be a containment mapping from $Q_1'$ to $Q_1$, contradicting the fact that $Q_1 \neq Q_1'$. 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, $Q'_1 \subseteq Q_j$, and $Q_1 \subseteq Q'_1 \subseteq Q_j$. Then $\mathcal{Q}_m$ cannot be minimal, since it has a redundant query $Q_1$. ∎

Theorem 5.2 gives an algorithm *UCQstable* for testing stability of UCQs, as shown in Figure 10.

**Algorithm UCQstable:** Test stability of unions of conjunctive queries
**Input:** • $\mathcal{Q}$: A finite union of conjunctive queries.
• $B$: Binding restrictions of the relations used in $\mathcal{Q}$.
**Output:** Decision about the stability of $\mathcal{Q}$.
**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 algorithm CQstable* to test the stability of $Q_i$;
• If $Q_i$ is not stable, then query $\mathcal{Q}$ is not stable.
(3) Query $\mathcal{Q}$ is stable.

Figure 10: Algorithm: UCQstable

## 5.2 Algorithm: UCQstable*

Similar to Theorem 3.2, we have the following theorem.

**Theorem 5.3** *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 $\mathcal{Q} = \mathcal{Q}_s$, 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 that $\mathcal{Q} \neq \mathcal{Q}_s$, i.e., 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 CQ's is transitive, there must be a 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 $Q$ in $\mathcal{Q} - \mathcal{Q}_s$ such that $Q_i \subset Q$ (i.e., $Q_i \subseteq Q$ but $Q_i \neq Q$). ($Q_i$ 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 query with the answerable subgoals of $Q_i$. Since $Q_i$ is not stable, by Theorem 5.3, $Q'_i \neq Q_i$. 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 5.2, to prove that $\mathcal{Q}$ is not stable, we need to prove 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 from $Q_i$, which can not be true by Theorem 3.2.
2. $t_h$ is from a query $Q_k$ in $\mathcal{Q}_s$. Then there is a symbol mapping from the variables of $Q_k$ to the obtainable tuples. This mapping also serves as a containment mapping from $Q_k$ to $Q'_i$. Therefore, $Q_i \subseteq Q'_i \subseteq Q_k$, contradicting the fact that $Q_i$ is not contained in any CQ in $\mathcal{Q}_s$.
3. $t_h$ is from a query $Q_j$ in $\mathcal{Q} - \mathcal{Q}_s$. Then $Q_i \subset Q'_i \subseteq Q_j$, contradicting to the fact that no query in $\mathcal{Q} - \mathcal{Q}_s$ can properly contain $Q_i$.

■

Theorem 5.3 gives another algorithm *UCQstable** for testing stability of UCQs, as shown in Figure 11. 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 restrictions 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 algorithm CQStable* to test the stability of $Q_i$;
  (b) If $Q_i$ is stable, add $Q_i$ to $\mathcal{Q}_s$;

(2) Test whether $\mathcal{Q} \subseteq \mathcal{Q}_s$ as queries;
(3) If $\mathcal{Q} \subseteq \mathcal{Q}_s$, then query $\mathcal{Q}$ is stable; otherwise, query $\mathcal{Q}$ is not stable..

Figure 11: Algorithm: UCQstable*

One corollary of Theorem 5.2 and Theorem 5.3 is that stability of bounded datalog queries [CGKV88] is decidable. That is because 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 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 its corresponding UCQ.

# 6 Stability of Conjunctive Queries with Arithmetic Comparisons

In this section we study stability of CQ's with arithmetic comparisons (CQAC's for short). Let $Q$ be a CQAC. Let $O(Q)$ be the set of ordinary (uninterpreted) subgoals of $Q$ that do not have comparisons. Let $\mathcal{C}(Q)$ 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 safe, i.e., each variable in the comparisons appears in some ordinary subgoal.

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 6.1** Consider the query

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

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

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

is an equivalent query with fewer subgoals. □

## 6.1 Answerable Subquery of a CQAC

**Definition 6.1 (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 includes the head of $Q$, the answerable subgoals of $Q$, and all the comparisons of the bound variables that can be derived from $\mathcal{C}(Q)$. □

The answerable subquery $Q_a$ of a CQAC $Q$ can be computed as follows. First derive all the equalities from $\mathcal{C}(Q)$. That is, if $Q$ contains equalities such as $X = Y$, or equalities that can be derived from inequalities (e.g., if we can derive $X \leq Y$ and $X \geq Y$, then we know $X = Y$), then we substitute variable $X$ by $Y$. Then 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 $\mathcal{C}(Q)$. $Q_a$ includes *all* the constraints of $\mathcal{V}$, because $\mathcal{C}(Q)$ may derive more constraints that $\mathcal{V}$ should satisfy than $\mathcal{C}(Q)$. For instance, assume variable $X$ is bound, and variable $Y$ is not. If $Q$ 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 $Q_a \subseteq Q$, 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 6.2** Consider query

$$P\texttt{:ans(Y) :- p(X) \& r(X,Y) \& r(A,B) \& A} < \texttt{B \& X} \leq \texttt{A \& A} \leq \texttt{Y}$$

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

$P_a$: `ans(Y) :- p(X) & r(X,Y) & X` $\leq$ `Y`

Using the GZO algorithm we know that $P_a \not\subseteq P$. Therefore, we may claim that query $P$ is not stable. However, actually query $P$ *is* stable. As we will see in Section 6.3, query $P$ is equivalent to the union of the following two queries. (Note that all the ordinary subgoals in these two queries are answerable.)

$T_1$: `ans(Y) :- p(X) & r(X,Y) & X` $<$ `Y`
$T_2$: `ans(Y) :- p(Y) & r(Y,Y) & r(Y,B) & Y` $<$ `B`

□

The reason the above "algorithm" fails is that, the only case where $P_a \not\subseteq P$ 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. This example suggests that we need to use the idea in [Klu88] to test stability of CQAC's. That is, we need to consider *all* the total orders of the variables in the query.[6]

## 6.2 Algorithm: CQAC1stable

Before we give the algorithm for testing the statiblity of any CQAC, we first consider the case where the above "algorithm" works. It turns out that the above "algorithm" is correct when a CQAC does not include comparisons $\{\leq, \geq, =\}$. Figure 12 shows an algorithm *CQAC1stable* that tests stability of CQAC's without nonstrict comparisons $\{\leq, \geq, =\}$, i.e., their comparisons can only have $\{<, >\}$.

**Algorithm CQAC1stable:** Test stability of CQAC's without comparisons $\{\leq, \geq, =\}$
**Input:** • $Q$: A CQAC without comparisons $\{\leq, \geq, =\}$.
• $B$: Binding restrictions 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 $\mathcal{C}(Q)$.
- Let $Q_a$ be the query with $\mathcal{A}$, $\mathcal{I}$, and the head of $Q$.

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

Figure 12: Algorithm: CQAC1stable

Before giving the proof of the correctness of the algorithm CQAC1stable, we review the following theorem from [GSUW94]:

**Theorem 6.1** *Let $Q_1$ and $Q_2$ be two CQAC's. Assume that no variable appears twice among their ordinary subgoals, and no constant appears in their ordinary subgoals. Let $O(Q_1)$ (resp. $O(Q_2)$) and $\mathcal{C}(Q_1)$ (resp. $\mathcal{C}(Q_2)$) be the ordinary subgoals and comparisons of query $Q_1$ (resp. $Q_2$), respectively. Then $Q_2 \subseteq Q_1$ if and only if the following holds. Let $H$ be the set of all containment mappings from $O(Q_1)$ to $O(Q_2)$. Then $H$ is nonempty, and $\mathcal{C}(Q_2)$ logically implies $\vee_{h \ in \ H} h(\mathcal{C}(Q_1))$.* □

**Theorem 6.2** *The algorithm CQAC1stable correctly decides the stability of a CQAC without comparisons $\{\leq, \geq, =\}$.* □

**Proof:** If query $Q_a$ is equivalent to query $Q$, clearly query $Q$ 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 $Q_a \not\subseteq Q$, 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

[6] 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 $X_i < X_j$.

$$
\begin{array}{ll}
Q' : H \text{ :- } O_1, \ldots, O_k, \ldots, O_n, C_1, \ldots, C_m & \Leftarrow \text{instantiation } f \text{ (database } D_2) \\
& \quad\quad\quad \uparrow \text{extend} \\
Q'_a : H \text{ :- } O_1, \ldots, O_k, C'_1, \ldots, C'_m & \Leftarrow \text{instantiation } s \text{ (database } D_1)
\end{array}
$$

Figure 13: Proof of the correctness of the algorithm CQAC1stable

tuples. Figure 13 shows the main idea of the construction. We first rewrite the queries $Q$ and $Q_a$ to queries $Q'$ and $Q'_a$ that satisfy the assumptions in Theorem 6.1. That is, no variable in $Q'$ (resp. $Q'_a$) appears twice among its ordinary subgoals, and no constant appears in its ordinary subgoals. The rewriting can be done as follows: (1) if a variable appears twice in the ordinary subgoals, we use distinct variables and equate them by arithmetic equality contraints; (2) we replace constants in the ordinary subgoals by new variables and equate those variables to the desired constants.

Assume that $Q'$ have $n$ ordinary subgoals, $O_1, \ldots, O_n$. Without loss of generality, let $O_1, \ldots, O_k$ be the subgoals corresponding to the answerable subgoals of $Q$. These $k$ subgoals are also all the ordinary subgoals of $Q'_a$. Since $Q_a \not\sqsubseteq Q$, we have $Q'_a \not\sqsubseteq Q'$. By Theorem 6.1, $\mathcal{C}(Q'_a)$ does not imply $\vee_{h \text{ in } H} h(\mathcal{C}(Q'))$, where $H$ includes all the containment mappings from $Q'$ to $Q'_a$. Then there must be an instantiation $s$ for the variables in $Q'_a$, such that $s(\mathcal{C}(Q'_a))$ is true, while no $h$ in $H$ can make $s\Big(h(\mathcal{C}(Q'))\Big)$ true.

Let database $D_1$ include tuples $t_1, \ldots, t_k$ under the instantiation $s$. Notice that: (1) $Q_a$ does not include comparisons $\{\leq, \geq, =\}$, and (2) the comparisons in $\mathcal{C}(Q'_a)$ 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 $Q$. 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 $\mu$ from $O(Q')$ to $O(Q'_a)$, such that $s\Big(\mu(\mathcal{C}(Q'))\Big)$ is true, contradicting to the fact that no $h$ in $H$ can make $s\Big(h(\mathcal{C}(Q'))\Big)$ true. ∎

The algorithm CQAC1stable also shows how to compute the complete answer to a stable CQAC $Q$ without comparisons $\{\leq, \geq, =\}$. For any database $D$, we compute $ANS(Q, D)$ by computing $ANS(Q_a, D)$. To compute $ANS(Q_a, D)$, we first using 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 supplementary relation that do not satisfy the comparisons $\mathcal{C}(Q_a)$.

**EXAMPLE 6.3** Consider query

$P$: `ans(B) :- p(B) & r(A,B) & r(A,C) & A < C & C < B`

where relation $p$ has a binding pattern $f$, and relation $r$ has a binding pattern $fb$. Since $P$ does not have comparisons $\{\leq, \geq, =\}$, we can 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 of the two variables is $A < C$. Thus the following is the answerable subquery of $Q$:

$P_a$: `ans(B) :- p(B) & r(A,B) & A < B`

We then test whether $P_a \subseteq P$. We rewrite the queries $P$ and $P_a$ to the following queries $P'$ and $P'_a$ that satisify the assumptions in Theorem 6.1.

$P'$: `ans(B) :- p(B) & r(A,X) & r(Y,C) & X = B & Y = A & A < C & C < B`
$P'_a$: `ans(B) :- p(B) & r(A,X) & X = B & A < B`

The comparisons in query $P'$ (i.e., $\mathcal{C}(P')$) are: $X = B$ & $Y = A$ & $A < C$ & $C < B$; the comparisons in query $P'_a$ (i.e., $\mathcal{C}(P'_a)$) are: $X = B$ & $A < B$. There is only one containment mapping $\mu$ from $P'$ to $P'_a$:

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

Thus we need to verify whether $\mathcal{C}(P'_a)$ logically implies $\mu(\mathcal{C}(P'))$:

$$X = B \wedge A < B \Rightarrow X = B \wedge A = A \wedge A < X \wedge X < B$$

That is:

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

which is false, and we have $P_a \not\subseteq P$. Therefore, query $P$ is not stable. 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 $ANS(P, D_1) = \{3\}$, and $ANS(P, D_2) = \phi$, 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" represents any constant that is between 1 and 3, but not equivalent to 1 and 3. □

### 6.3 Algorithm: CQACstable

Now we show how to test stability of CQAC's by giving the following theorem.

**Theorem 6.3** *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 $Q^\lambda$ be the corresponding query that includes the ordinary subgoals of $Q$ and all the inequalities and equalities of this order $\lambda$. Then query $Q$ is stable if and only if for all $\lambda \in \Omega(Q)$, query $Q^\lambda_a \subseteq Q$, where $Q^\lambda_a$ is the answerable subquery of $Q^\lambda$.* □

$$Q = \bigcup \left\{ \begin{array}{ccccc} Q^{\lambda_1} & \subseteq & Q^{\lambda_1}_a & \subseteq & Q \\ Q^{\lambda_2} & \subseteq & Q^{\lambda_2}_a & \subseteq & Q \\ \vdots & & & & \\ Q^{\lambda_m} & \subseteq & Q^{\lambda_m}_a & \subseteq & Q \end{array} \right.$$

Figure 14: Proof of Theorem 6.3, "*If*" part

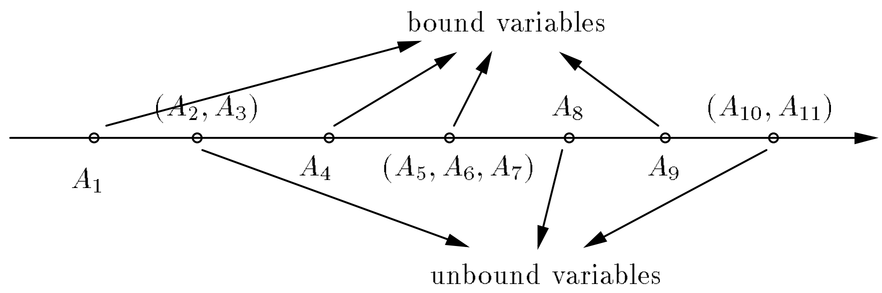


Figure 15: Proof of Theorem 6.3, "*Only If*" part

**Proof:** *If*: Assume that for each $\lambda_i \in \Omega(Q)$, $Q_a^{\lambda_i} \subseteq Q$. As shown in Figure 14, $Q = \bigcup_{\lambda \in \Omega(Q)} Q^{\lambda}$. In addition, for each $\lambda_i \in \Omega(Q)$, $Q^{\lambda_i} \subseteq Q_a^{\lambda_i}$. Then we have $Q = \bigcup_{\lambda \in \Omega(Q)} Q^{\lambda} \subseteq \bigcup_{\lambda \in \Omega(Q)} Q_a^{\lambda} \subseteq Q$. Thus $Q = \bigcup_{\lambda \in \Omega(Q)} Q_a^{\lambda}$. 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 $\lambda_1$, such that $Q_a^{\lambda_1} \not\subseteq Q$. Figure 15 shows the main idea of a total order. The variables on the left side must be smaller than the variables on the fight side. Some variables must 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 (i.e., variables $A_1$, $A_4, A_5, A_6, A_7, A_9$ in the figure) are bound by the answerable subgoals. Notice that we need to consider the equalities to compute all the answerable subgoals given the binding restrictions of the relations. That is because these equalities can help bind more variables.

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 essential the same as the construction in the proof of Theorem 6.2. That is, let $s$ be the instantiation of the variables in $Q_a^{\lambda_1}$ that makes $\mathcal{C}(P_a^{\lambda_1}) \Rightarrow \mathcal{C}(P)$ false, where $P_a^{\lambda_1}$ and $P$ are the rewritten queries of $Q_a^{\lambda_1}$ and $Q$ that satisfy the assumptions in Theorem 6.1. These variables correspond to the bound variables in the total order $\lambda_1$. Let $D_1$ include the tuples under the instantiation $s$.

Since $Q_a^{\lambda_1} \not\subseteq Q$, $Q_a^{\lambda_1}$ 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 $D_1$), since we chose new distinct values for the unbound variables. Following the same idea in the proof of Theorem 6.2, 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 6.3 gives an algorithm *CQACstable* (shown in Figure 16) that tests the stability of any CQAC, even if the query has comparisons $\{\leq, \geq, =\}$. The algorithm considers all the total orders of the variables, including those with equalities.

**Algorithm CQACstable:** Test stability of CQAC's
**Input:** • $Q$: A conjunctive query with arithmetic comparisons.
• $B$: Binding restrictions 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 $\lambda \in \Omega(Q)$:
- Compute the answerable subquery $Q_a^\lambda$ of query $Q^\lambda$;
- Test $Q_a^\lambda \subseteq Q$ by calling the GZO algorithm;
- If $Q_a^\lambda \not\subseteq Q$, then query $Q$ is not stable.

(3) Query $Q$ is stable.

Figure 16: 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 $Q^\lambda$, we compute $ANS(Q_a^\lambda, D)$, where $Q_a^\lambda$ is the answerable subquery of $Q^\lambda$. Since $Q_a^\lambda$ 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 $Q_a^\lambda$. We take the union of the answers for all the total orders in $\Omega(Q)$ as $ANS(Q, D)$.

**EXAMPLE 6.4** Consider the query $P$ in Example 6.2. It has the following 8 total orders:

$\lambda_1$: $X < A = Y < B$
$\lambda_2$: $X < A < Y < B$
$\lambda_3$: $X < A < Y = B$
$\lambda_4$: $X < A < B < Y$
$\lambda_5$: $X = A = Y < B$
$\lambda_6$: $X = A < Y < B$
$\lambda_7$: $X = A < Y = B$
$\lambda_8$: $X = A < B < Y$

For each of total order $\lambda_i$, we can write its corresponding query $P^{\lambda_i}$. Here are all the 8 queries:

$P^{\lambda_1}$: ans(Y) :- p(X) & r(X,Y) & r(Y,B) & X < Y & Y < B
$P^{\lambda_2}$: ans(Y) :- p(X) & r(X,Y) & r(A,B) & X < A & A < Y & Y < B
$P^{\lambda_3}$: ans(Y) :- p(X) & r(X,Y) & r(A,Y) & X < A & A < Y
$P^{\lambda_4}$: ans(Y) :- p(X) & r(X,Y) & r(A,B) & X < A & A < B & B < Y
$P^{\lambda_5}$: ans(Y) :- p(Y) & r(Y,Y) & r(Y,B) & Y < B
$P^{\lambda_6}$: ans(Y) :- p(X) & r(X,Y) & r(X,B) & X < Y & Y < B
$P^{\lambda_7}$: ans(Y) :- p(X) & r(X,Y) & r(X,Y) & X < Y
$P^{\lambda_8}$: ans(Y) :- p(X) & r(X,Y) & r(X,B) & X < B & B < Y

For each total order $\lambda_i$, we construct its corresponding answerable subquery $P_a^{\lambda_5}$. The following are the 8 answerable subqueries:

$P_a^{\lambda_1}$: ans(Y) :- p(X) & r(X,Y) & r(Y,B) & X < Y & Y < B
$P_a^{\lambda_2}$: ans(Y) :- p(X) & r(X,Y) & X < Y
$P_a^{\lambda_3}$: ans(Y) :- p(X) & r(X,Y) & X < Y
$P_a^{\lambda_4}$: ans(Y) :- p(X) & r(X,Y) & X < Y
$P_a^{\lambda_5}$: ans(Y) :- p(Y) & r(Y,Y) & r(Y,B) & Y < B
$P_a^{\lambda_6}$: ans(Y) :- p(X) & r(X,Y) & r(X,B) & X < Y & Y < B
$P_a^{\lambda_7}$: ans(Y) :- p(X) & r(X,Y) & X < Y
$P_a^{\lambda_8}$: ans(Y) :- p(X) & r(X,Y) & r(X,B) & X < B & B < Y

Each of the 8 answerable subquery can be proved to be contained in $P$. Therefore, query $P$ is stable. Actually, if we combine queries $P^{\lambda_1}$, $P_a^{\lambda_2}$, $P^{\lambda_3}$, $P^{\lambda_4}$, $P^{\lambda_6}$, $P_a^{\lambda_7}$, $P_a^{\lambda_8}$, and we have query

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

and its answerable subquery is:

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

which is the query $T_1$ in Example 6.2. In addition, $P_a^{\lambda_5}$ is equivalent to the query $T_2$ in the example. We can prove that $P_a^{\lambda_{1,2,3,4,6,7,8}} \subseteq P^{\lambda_{1,2,3,4,6,7,8}}$. Since query $P = P^{\lambda_{1,2,3,4,6,7,8}} \cup P^{\lambda_5}$, we have $P = T_1 \cup T_2$. □

# 7 Stability of datalog Queries

In this section we study stability of datalog queries, i.e., Horn-clause programs without function symbols. We show that if a datalog query has a feasible rule/goal graph, then the query is stable.

**EXAMPLE 7.1** 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:

$r_1$: `reachable(X,Y) :- flight(X,Y)`
$r_2$: `reachable(X,Y) :- flight(X,Z),reachable(Z,Y)`

Let queries $P_1$ and $P_2$ be $reachable(sfo, X)$ and $reachable(X, sfo)$, respectively. That is, query $P_1$ asks for all the airports that are reachable from the airport $sfo$, while query $P_2$ asks for all the airports from which the airport $sfo$ is reachable. Although we cannot retrieve all the flight facts, the answer to query $P_1$ 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 $P_1$. That is because 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 $i = 1, \ldots, n-1$, $<a_i, a_{i+1}>$ is a tuple in the *flight* relation. Therefore, airport $a_i$ can be retrieved in the $(i-1)$st step during the computation above.

For query $P_2$, we cannot compute its answer in the same way as $P_2$, 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 $P_2$ at all, since there can always be an airport from which $sfo$ is reachable, but this airport cannot be retrieved from the relation. □

## 7.1 Rule/goal Graphs

We assume that the reader has the background of rule/goal graphs. (Details are found in Chapter 12 in [Ull89].) Given a set of rules and a query goal $p^\alpha$ with an adornment $\alpha$ (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.

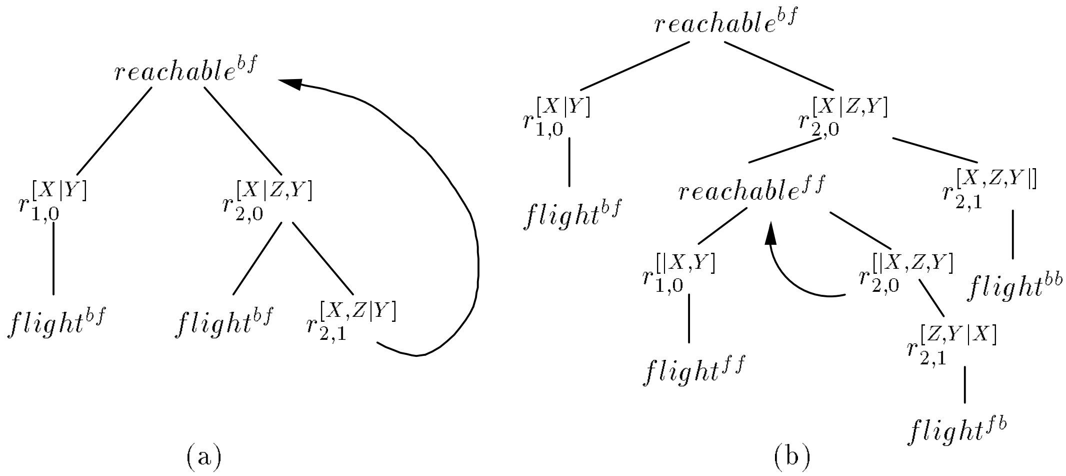


Figure 17: Two RGG's for the query goal $reachable^{bf}$

**EXAMPLE 7.2** Consider the query $P_1$ in Example 7.1. 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 17(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 $Y$ and $Z$ 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 17(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 17(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 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 that has the unique binding pattern property with respect to the query goal.

## 7.2 Feasible Rule/goal Graphs

Given a set of rules on EDB relations with binding restrictions, an RGG of a goal node $p^{\alpha}$ is *feasible* if all its EDB goals in the RGG use only the adornments that are permitted by the EDB relations. For instance, in Example 7.1, the RGG in Figure 17(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 17(b) is not feasible, since it has two EDB goals ($flight^{ff}$ and $flight^{fb}$) that are not permitted by the $flight$ relation.

**Theorem 7.1** *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 $p^{\alpha}$ 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 7.1 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].

## 7.3 What If a Feasible RGG Does not Exist

In some cases, even though a set of rules do 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. The following example is a case in point.

**EXAMPLE 7.3** 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 $P_1$ , since the variable $W$ in the third subgoal $flight(W, Z)$. However, the new rules are equivalent to the old ones, and query $P_1$ on the new rules is still stable. □

Example 7.3 shows a similar phenomenon as CQ's; 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 CQ's and UCQ's. Shmueli [Shm93] showed that for a datalog program $P$, whether $P$ is equivalent to datalog program $P'$, where $P'$ is produced by removing a subgoal from a rule of $P$, is undecidable.

**Theorem 7.2** *Stability of datalog programs is undecidable.* □

**Proof:** Let $P_1$ and $P_2$ be two arbitrary datalog queries. We show that a decision procedure for the stability of datalog programs would allow us to decide whether $P_1 \subseteq P_2$. Since containment of datalog programs is undecidable, we prove the claim.[7] 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 $P_1$ and $P_2$, named $p_1$ and $p_2$ respectively, have arity $m$. Let $\mathcal{Q}$ be the datalog query consisting of all the rules in $P_1$ and $P_2$, 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 $P_1 \subseteq P_2$ if and only if query $\mathcal{Q}$ is stable.

"Only If": Assume $P_1 \subseteq P_2$. Hence $\mathcal{Q} = P_2$. Since the EDB relations in $P_2$ can return all their tuples for free, $P_2$ (thus $\mathcal{Q}$) is stable.

"If": Assume $P_1 \not\subseteq P_2$, we prove that query $\mathcal{Q}$ cannot be stable. Since $P_1$ is not contained in $P_2$, there exists a database $D$ of the EDB relations in $P_1$ and $P_2$, such that $ANS(P_1, D) \not\subseteq ANS(P_2, D)$. That is, there is a tuple $t \in ANS(P_1, D)$, while $t \notin ANS(P_2, D)$. 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 $P_1$ and $P_2$. 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 $ANS(P_2, D)$ for both $D_1$ and $D_2$. For $D_1$, since relation $e$ is empty, $ANS(\mathcal{Q}, D_1) = ANS(P_2, D)$, which does not include tuple $t$. However, for $D_2$, relation $e$ has a tuple $\langle z \rangle$, and $ANS(\mathcal{Q}, D_1) = ANS(P_1, D) \cup ANS(P_2, D)$, which includes tuple $t$. Therefore, $ANS(\mathcal{Q}, D_1) \neq ANS(\mathcal{Q}, D_2)$, and query $\mathcal{Q}$ is not stable. ■

# 8 Conclusion

In this paper we studied fundamental problems of answering queries in the presence of binding restrictions: can the complete answer to a query be computed given the restrictions? If so, how to compute it? We first study conjunctive queries, and show that a conjunctive query is stable if and only if its minimal equivalent query $Q_m$ has a feasible order (an order in which each subgoal is queried with a

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

legal binding pattern) of all the subgoals in $Q_m$. We propose two algorithms for testing stability of conjunctive queries, and we prove this problem is $\mathcal{NP}$-complete.

For a nonstable conjunctive query, whether its complete answer can be computed is data dependent. We propose a decision tree that guides the query planning process to compute the complete answer to a conjunctive query, if it can be computed at all. Two planning strategies — a pessimistic strategy and an optimistic strategy — can be taken while traversing the decision tree. We also study stability of unions of conjunctive queries, and conjunctive queries with arithmetic comparisons. In both cases we propose algorithms for testing stability of queries. Finally, we study datalog queries, and prove that if a set of rules and a query goal have a feasible rule/goal graph, then the query is stable. However, stability of datalog programs is undecidable.

**Acknowledgments:** The author thanks Jeff Ullman for his valuable comments on this material.

## 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.

[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 *Proc. of ACM Symposium on Principles of Database Systems (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. *ACM Symposium on Theory of Computing (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 Y. Levy. Recursive plans for information gathering. *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, IJCAI-97*, 1997.

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

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

[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 *Proc. of ACM Symposium on Principles of Database Systems (PODS)*, pages 45–55. ACM Press, 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 *Proc. of ACM 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.

[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. *International Conference on Data Engineering (ICDE)*, 2000.

[LMSS95] Alon Y. Levy, Alberto O. Mendelzon, Yehoshua Sagiv, and Divesh Srivastava. Answering queries using views. In *Proc. of ACM Symposium on Principles of Database Systems (PODS)*, pages 95–104, 1995.

[LRO96] Alon Y. 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 *Proc. of ACM 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 *Proc. of ACM SIGMOD*, pages 544–546, 1998.

[Mor88] Katherine A. Morris. An algorithm for ordering subgoals in NAIL! In *Proc. of ACM Symposium on Principles of Database Systems (PODS)*, pages 82–88. ACM Press, 1988.

[NS87] Jeffrey F. Naughton and Yehoshua Sagiv. A decidable class of bounded recursions. In *Proc. of ACM Symposium on Principles of Database Systems (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. *International Conference on Data Engineering (ICDE)*, pages 48–55, 1996.

[RSU95] Anand Rajaraman, Yehoshua Sagiv, and Jeffrey D. Ullman. Answering queries using templates with binding patterns. In *Proc. of ACM Symposium on Principles of Database Systems (PODS)*, pages 105–112, 1995.

[Sag85] Yehoshua Sagiv. On computing restricted projections of representative instances. In *Proc. of ACM Symposium on Principles of Database Systems (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 *Proc. of ACM 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 *Proc. of ACM Symposium on Principles of Database Systems (PODS)*, pages 74–81. ACM Press, 1988.

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

[YLUGM99] Ramana Yerneni, Chen Li, Jeffrey D. Ullman, and Hector Garcia-Molina. Optimizing large join queries in mediation systems. *International Conference on Database Theory (ICDT)*, pages 348–364, 1999.

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