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

Chen Li
Department of Computer Science, Stanford University, CA 94305
Tel: (650)725-7359, Fax: (650)725-2588, Email: 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 develop algorithms for testing stability of conjunctive queries, unions of conjunctive queries, and conjunctive queries with arithmetic comparisons. For a nonstable conjunctive query, we develop a decision tree that guides the query planning process to compute its complete answer, if it can be computed at all. Finally, we prove that stability of datalog programs is undecidable, and give a sufficient condition for stability of datalog programs.

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

1 Introduction

The goal of information-integration systems (e.g., [2, 10, 12, 13, 14, 15, 22, 23, 25]) is to support seamless access to heterogeneous data sources. In these systems, sources may have limited access patterns, also called binding patterns. That is, the binding patterns of the sources require values to be specified for certain attributes in order to retrieve data from a relation. For instance, many Web movie sources like The IMDB [11] and Cinemachine [5] 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. These sources do not accept queries such as "return all the movie information you know about."

In this study we consider the problem of computing the complete answer to a query when there is limited access to relations. The *complete* answer to a query is the answer that could be computed if we could retrieve all the tuples from the relations mentioned in the query. The following example shows that, nevertheless, in some cases the complete answer to a query can be computed.

EXAMPLE 1.1 Suppose that 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

---

*Due to space limitations, we have not provided all the proofs of the lemmas and theorems in this paper. They are available in the full version of the paper [18].

[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).

$s(Movie, Award)$ has binding pattern bf. Consider the following query that asks for the awards of the movies in which Henry Fonda starred:

$$Q_1: \ \texttt{ans(A) :- r('Henry Fonda', M), s(M, A)}$$

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. Although only part of the two relations was retrieved, 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. 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 (details in Section 3). In addition, the following example shows that a query can be stable even if such a feasible order does not exist.

EXAMPLE 1.2 Suppose relation r has a binding pattern bff, and relation s has a binding pattern fbb. Consider query:

$$Q_2: \ \texttt{ans(V, X) :- r(a, V, Y), r(b, W, U), s(X, V, W), s(X, Z, W)}$$

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:

$$Q_2': \ \texttt{ans(V, X) :- r(a, V, Y), r(b, W, U), s(X, V, W)}$$

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 plan following this order can compute the complete answer, which is also the complete answer to Q_2. □

Example 1.2 suggests that testing stability of a CQ is more than just checking the existence of a feasible order of all its subgoals. In this paper we study how to test stability of queries. We will see this problem is closely related to containment between queries.

Major Results We consider stability of a variety of queries. The following are the main results:

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 develop a decision tree (Figure 1 in Section 4) to guide the planning process to compute the complete answer to a CQ. Two planning strategies can be taken while traversing the decision tree.
3. We study stability of finite 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 formed from a subset of its own subgoals. We propose an algorithm for testing stability of CQAC's.
5. We prove that stability of datalog programs is undecidable. We also show that if a datalog query has a feasible rule/goal graph with respect to its query goal, then the query is stable.

2 Preliminaries

2.1 Source relations with binding restrictions

Let $r_1, \ldots, r_n$ be n source relations. Each relation has a set of binding patterns representing the possible queries that the relation can accept. In each binding pattern, an attribute is adorned as b (a value must be specified for this attribute) or f (the attribute can be free). For instance, a relation $r(A, B, C)$ with the binding patterns $\{bff, ffb\}$ requires that every query to the relation must either supply a value for the first argument, or supply a value for the third argument. Let $\mathcal{R}$ denote the relations with their adornments ("source descriptions" for short). A conjunctive query (CQ) is denoted by:

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

In each subgoal $g_i(\bar{X}_i)$, predicate g_i is a relation in $\mathcal{R}$, and every argument in the subgoal is either a variable or a constant. The variables $\bar{X}$ in the head 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.

2.2 Query stability

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. □

However, we may not be able to retrieve all the tuples from a relation due to its binding restrictions. 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. These tuples are not retrieved because we did not obtain the necessary bindings.* □

Assume 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. This approach would not terminate, since there will be an infinite number of strings that need to be tested. Instead, we assume that each binding we use is either from the user query, or from the result of a source query, while the binding is from the appropriate domain.

Definition 2.2 (**legal plan**) A plan for a query is *legal* if the plan sends queries to relations with legal binding patterns. □

In the rest of this paper, unless otherwise specified, the term "a plan for a query" means a legal plan for the query.

Definition 2.3 (**exhaustive plan**) Let Q be a query on a set of relations $\mathcal{R}$. The *exhaustive plan* for Q is a plan that accesses all the possible relations $\mathcal{R}$ to obtain bindings, and retrieves as many tuples from the relations as possible to answer the query. [7, 20] showed how to implement an exhaustive plan by translating the query Q and the relations $\mathcal{R}$ with their binding patterns into a datalog program, and evaluating the program on the relations $\mathcal{R}$. □

Definition 2.4 (**observable tuples**) Given a database of relations $\mathcal{R}$, a tuple t of a relation $r_i \in \mathcal{R}$ is *observable* if it can be retrieved by the exhaustive plan for Q. That is, we can retrieve t from r_i

starting with the initial bindings in the query, and perhaps using several preliminary queries to other relations in order to obtain the necessary bindings to retrieve t from r_i. □

Definition 2.5 (**stable query**) A query on relations with binding restrictions is *stable* if there exists a legal plan for the query that computes the query's complete answer independent of the database of the relations. □

In general, if we want to prove that a query Q is not stable, we need to show two databases D_1 and D_2, such that they have the same observable tuples. However, they yield different answers to Q, i.e., $ANS(Q, D_1) \neq ANS(Q, D_2)$. Therefore, even if we execute the exhaustive plan, we still cannot tell whether the answer computed using the observable tuples is the complete answer or not.

3 Stability of CQ's and UCQ's

3.1 Stability of CQ's

Definition 3.1 (**feasible order of subgoals**) Some subgoals $g_1(\bar{X}_1), \ldots, g_k(\bar{X}_k)$ in a CQ form a *feasible order* if each subgoal $g_i(\bar{X}_i)$ in the order is *executable*; that is, there is a binding pattern p of the relation g_i, such that for each argument A in $g_i(\bar{X}_i)$ that is adorned as b in p, either A is a constant, or A appears in a previous subgoal. A CQ is *feasible* if it has a feasible order of *all* its subgoals. □

Query Q_2' in Example 1.2 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. Query Q_2 is not feasible, since it does not have such a feasible order. The example also shows that two equivalent queries (i.e., Q_2 and Q_2') can have different feasibility.

Lemma 3.1 *A feasible CQ is stable.* □

Proof: (Sketch) Assume Q is a feasible CQ on relations with binding restrictions. That is, Q has a feasible order $g_1(\bar{X}_1), \ldots, g_n(\bar{X}_n)$ of all its subgoals. For any database D, we can compute $ANS(Q, D)$ by executing the following *linear plan*. Compute the corresponding sequence of n *supplementary relations* [1, 30] $I_1, \ldots, I_n$, where I_i is the supplementary relation after the first i subgoals have been processed. Return the supplementary relation I_n as $ANS(Q, D)$. ■

As shown in Example 1.2, 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 [3]. From Lemma 3.1 we have the following corollary.

Corollary 3.1 *If a CQ has a minimal equivalent that is feasible, then the query is stable.* □

Theorem 3.1 *A CQ is stable iff its minimal equivalent is feasible.* □

Before we give the proof, we first define answerable subgoals of a CQ.

Definition 3.2 (**answerable subgoals**) A subgoal in a CQ is *answerable* if it is in a feasible order of some subgoals in the query. □

The answerable subgoals of a CQ can be computed by a greedy algorithm, called the *Inflationary* algorithm. That is, initialize the set Φ_a of answerable subgoals to be empty. With the variables bound by the subgoals in Φ_a, whenever a subgoal becomes executable by its relation, add this subgoal to Φ_a. Repeat this process until no more subgoals can be added to Φ_a. Clearly a query is feasible if and only if all its subgoals are answerable. Now we give the proof of Theorem 3.1.

Proof: (Sketch) Corollary 3.1 is the "If" part. Conversely, assume that a CQ Q has a minimal equivalent Q_m that is not feasible. We construct two databases, D_1 and D_2, such that $ANS(Q_m, D_1) \neq ANS(Q_m, D_2)$, but D_1 and D_2 have the same observable tuples. Instantiate each variable in Q_m with a distinct constant. Let D_1 include the tuples for all the subgoals in Q_m under this instantiation, while D_2 includes only the tuples for the answerable subgoals in Q_m. ∎

3.2 Two algorithms: CQstable and CQstable*

Theorem 3.1 gives an algorithm *CQstable* for testing the stability of a CQ Q as follows. First compute its minimal equivalent Q_m by deleting its redundant subgoals. Then use the Inflationary algorithm to test the feasibility of Q_m. If Q_m is feasible, then Q is stable; otherwise, Q is not stable. The complexity of the algorithm CQstable is exponential, since we need to minimize the CQ first, which is known to be $\mathcal{NP}$-complete [3]. There is a more efficient algorithm that is based on the following theorem.

Theorem 3.2 *Let Q be a CQ on relations with binding restrictions and Q_a be the query with the head of Q and the answerable subgoals of Q. Then Q is stable iff $Q = Q_a$.* □

Theorem 3.2 gives another algorithm *CQstable** for testing the stability of a CQ Q as follows. First compute all the answerable subgoals of Q. If all the subgoals are answerable, then Q is stable, and we do not need to test any query containment. Otherwise, construct query Q_a with these answerable subgoals and the head of Q. Test whether there is a containment mapping from Q to Q_a. If so, Q is stable; otherwise, Q is not stable. The algorithm CQstable* has two advantages: (1) If all the subgoals of Q are answerable, then we do not need to test whether $Q_a \subseteq Q$, thus its complexity is polynomial in this case. (2) As we will see in Section 5, this algorithm can be generalized to test stability of CQAC's.

3.3 Complexity of testing stability of CQ's

Theorem 3.3 *The problem of testing stability of CQ's is $\mathcal{NP}$-complete.* □

Proof: (Sketch) 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 [3]. We reduce this problem to the problem of testing the stability of a CQ. Let Q be $ans(\bar{X})$:- $g_1(\bar{X}_1), \ldots, g_n(\bar{X}_n)$, and Q' be $ans(\bar{X})$:- $g_1(\bar{X}_1), \ldots, g_k(\bar{X}_k)$, where $k < n$. We construct a query P on relations with binding restrictions, such that P is stable iff $Q' \subseteq Q$. Let A and B be two new variables that do not appear in the subgoals of Q. For each relation g_i, introduce a new relation g'_i with one more attribute than g_i, and g'_i has only one binding pattern $bff\ldots f$. Introduce a new monadic (i.e., 1-ary) relation h with a binding pattern f. Let P be the query with the same head of Q and subgoals $h(A), g'_1(A, \bar{X}_1), \ldots, g'_k(A, \bar{X}_k)$, $g'_{k+1}(B, \bar{X}_{k+1}), \ldots, g'_n(B, \bar{X}_n)$. By the construction of the relations h and $g'_1, \ldots, g'_n$, the answerable subgoals of P are $h(A), g'_1(A, \bar{X}_1), \ldots, g'_k(A, \bar{X}_k)$. Let P' be the query with these answerable subgoals. By Theorem 3.2, P is stable iff $P = P'$, i.e., there is a containment mapping ν from P to P'. It is easy to show that ν exists iff there is a containment mapping 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. ∎

3.4 Stability of UCQ's

Let $\mathcal{Q} = Q_1 \cup \cdots \cup Q_n$ be a UCQ, and all its CQ's have a common head predicate. There is a unique minimal subset of $\mathcal{Q}$ that is its minimal equivalent [28].

EXAMPLE 3.1 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 CQ's:

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. □

In analogy with the results for CQ's, we give the following two theorems:

Theorem 3.4 *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.* □

Theorem 3.5 *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.* □

Theorem 3.4 gives an algorithm *UCQstable* for testing stability of a UCQ $\mathcal{Q}$ as follows. Compute the minimal equivalent $\mathcal{Q}_m$ of $\mathcal{Q}$, and test the stability of each CQ in $\mathcal{Q}_m$ using the algorithm CQstable or CQstable*. If each CQ in $\mathcal{Q}_m$ is stable, query $\mathcal{Q}$ is stable; otherwise, $\mathcal{Q}$ is not stable. Theorem 3.5 gives another algorithm *UCQstable** for testing stability of a UCQ $\mathcal{Q}$ as follows. Compute all the stable queries of $\mathcal{Q}$ by calling the algorithms CQstable or CQstable*. If all the queries are stable, then $\mathcal{Q}$ is stable. Otherwise, let $\mathcal{Q}_s$ be the union of these answerable queries. Test whether $\mathcal{Q} \subseteq \mathcal{Q}_s$, i.e., $\mathcal{Q}$ is contained in $\mathcal{Q}_s$ as queries. If so, $\mathcal{Q}$ is stable; otherwise, it is not stable. The advantage of this algorithm is that we do not need to test whether $\mathcal{Q} \subseteq \mathcal{Q}_s$ if all the queries in $\mathcal{Q}$ are stable.

4 Nonstable CQ's

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.

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 of the complete answer 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. Attributes $A, B, \ldots, E$ have different domains. Consider the following two queries:

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

The two queries have the same subgoals but different heads. The two queries are not stable. However, we can still try to answer query Q_1 as follows: send a query $r(a, X, Y)$ to relation r. Assume we obtain three tuples: $\langle a, b_1, c_1\rangle$, $\langle a, b_2, c_2\rangle$, and $\langle a, b_2, c_3\rangle$. The supplementary relation $I_1 = \{\langle b_1, c_1\rangle, \langle b_2, c_2\rangle, \langle b_2, c_3\rangle\}$. 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.

However, we may still query relation s to retrieve tuples by using the D bindings provided by p, although relation p is not mentioned in the query, If the retrieved tuples include $\langle c_1, d_1\rangle$ and $\langle c_2, d_2\rangle$, we can still claim that the complete answer is $\{b_1, b_2\}$. On the other hand, if only one tuple $\langle c_2, d_2\rangle$ is retrieved from s, then we do not know whether $\langle b_2\rangle$ is the complete answer or not, since we do not know whether relation s has a tuple $\langle c_1, d\rangle$ (for some d value) that has not been retrieved. This tuple can join with the tuple $\langle b_1, c_1\rangle$ in I_1 to produce value b_1 as an answer.

Similarly, we can try to answer query Q_2. After the first subgoal is solved, the supplementary relation $I_1 = \{\langle b_1, c_1\rangle, \langle b_2, c_2\rangle, \langle b_2, c_3\rangle\}$. However, no matter what tuples are retrieved from relation s, we can never know the complete answer to Q_2, since there can always be a tuple $\langle c_1, d'\rangle$ in relation s that has not been retrieved, and d' is in the complete answer to Q_2. For both queries Q_1 and Q_2, if the supplementary relation $I_1 = \phi$, then their complete answers are both empty. □

An important observation on these two queries is that Q_1's distinguished variable B can be bound by the answerable subgoal $r(a, B, C)$, while Q_2's 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 Inflationary algorithm to find all its answerable subgoals Φ_a. If all the distinguished variables can be bound by Φ_a, we use a linear plan of a feasible order of Φ_a to compute their supplementary relation I_a. 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 the exhaustive plan for Q to retrieve tuples for the nonanswerable subgoals Φ_{na}.
   (a) If for every tuple $t^P \in I_a^P$, there is a tuple $t_a \in I_a$, such that the projection of t_a onto the distinguished variables is t^P, and t_a can join with some tuples for all the subgoals Φ_{na} (tuple t^P is 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 Φ_a, then the complete answer cannot be computed, unless $I_a = \phi$. To summarize, whether the complete answer to a nonstable CQ is computable is data dependent, since it is not known until some plan is executed.

4.2 The decision tree

We develop a decision tree (as shown in Figure 1) 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 1). Then we test the feasibility of the query Q_m by calling the Inflationary algorithm; that is, we test whether Q_m has a feasible order of all its subgoals. If so (arc 2 in Figure 1), Q_m (thus Q) is stable, and its answer can be computed by a linear plan following a feasible order of all the subgoals in Q_m.

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

1. If all the distinguished variables are bound by the subgoals Φ_a (arc 4), then the complete answer

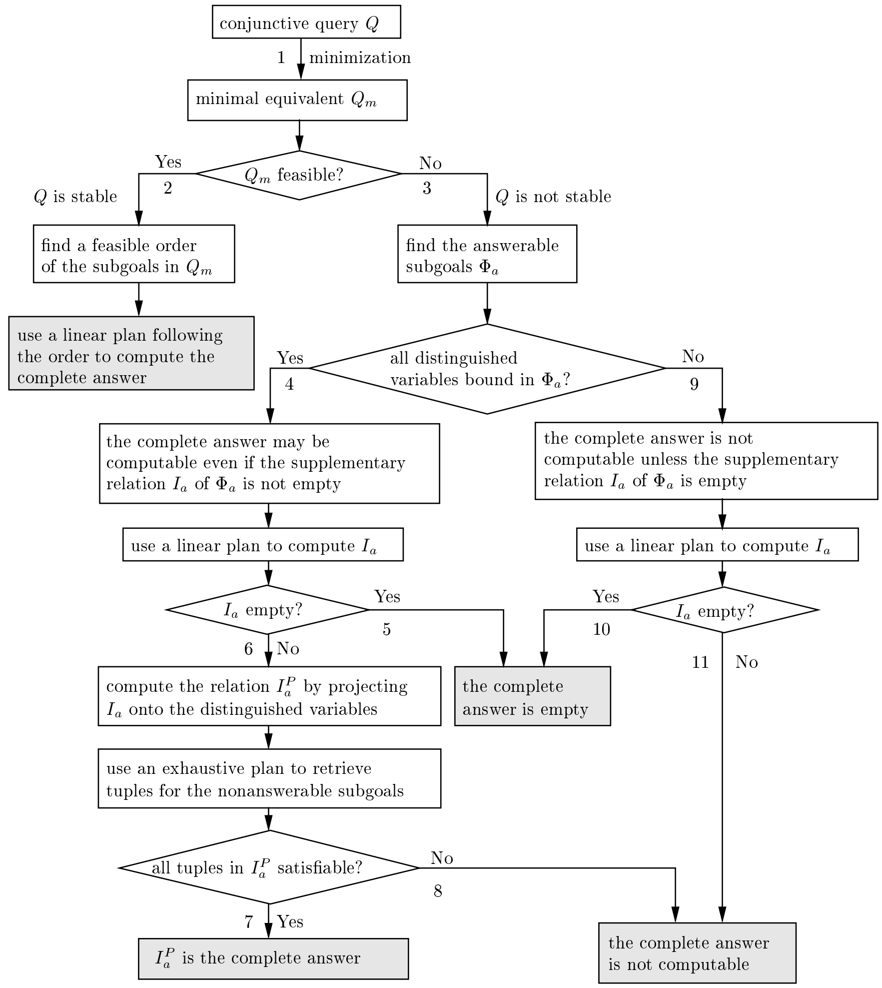

Figure 1: The decision tree of computing the complete answer to a CQ

to the query may be computed even if the supplementary relation I_a of subgoals Φ_a is not empty. We compute I_a by a linear plan following a feasible order of Φ_a.

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

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

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

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 taking the corresponding operations.

5 Stability of conjunctive queries with arithmetic comparisons

Assume Q is 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 subgoals of Q 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. In addition, all the comparisons are safe, i.e., each variable in the comparisons appears in some ordinary subgoal.

To test the stability of a CQAC, we might be tempted to generalize the algorithm CQstable. However, a CQAC may not have a minimal equivalent that has a subset of its subgoals, as shown by Example 14.8 in [30].

Definition 5.1 (answerable subquery of a CQAC) The *answerable subquery* of a CQAC Q on relations with binding restrictions, denoted by Q_a, is the query including 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. We first compute all the answerable ordinary subgoals $\mathcal{A}(Q)$ of query Q using the algorithm Inflationary. Note that 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 $X = Y$), we need to substitute variable X by Y before using the algorithm Inflationary to find all the answerable subgoals. Let $\mathcal{V}$ be the set of all the bound variables in $\mathcal{A}(Q)$. Derive all the inequalities 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. 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 algorithm in [9, 33] ("the GZO algorithm" for short). However, the following example shows that this "algorithm" does not always work.

EXAMPLE 5.1 Consider query

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

where relation p has a binding pattern f, and relation r has a binding pattern bf. Its answerable subquery is

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

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

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

□

The above "algorithm" fails because 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 $A = X = Y$, and the subgoal $r(A,B)$ becomes

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

Figure 2: Proof of Theorem 5.2

Algorithm CQACstable: Test stability of CQAC's
Input: • Q: A CQAC.
• 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^λ of query Q^λ;
• Test $Q_a^\lambda \subseteq Q$ by calling the GZO algorithm;
• If $Q_a^\lambda \not\subseteq Q$, claim that query Q is not stable.
(3) Claim that query Q is stable.

Figure 3: Algorithm: CQACstable

answerable. This example suggests that we need to consider *all* the total orders of the variables in the query, similar to the idea in [16].[2] Before we give the algorithm for testing the stability of any CQAC, we first show that the above "algorithm" is correct when a CQAC does not include comparisons $\leq$, $\geq$, or $=$ that are not strict.

Theorem 5.1 *If Q is a CQAC without comparisons $\{\leq, \geq, =\}$, then Q is stable iff $Q_a \subseteq Q$.* □

EXAMPLE 5.2 Consider query

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

where relation p has a binding pattern f, and relation r has a binding pattern fb. P does not have nonstrict comparisons, and its answerable subquery:

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

Using the GZO algorithm we can see that $P_a \not\subseteq P$. Therefore, query P is not stable. □

Theorem 5.2 *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 $\lambda \in \Omega(Q)$, let Q^λ be the corresponding query that includes the ordinary subgoals of Q and all the inequalities and equalities of this total order λ. Query Q is stable if and only if for all $\lambda \in \Omega(Q)$, $Q_a^\lambda \subseteq Q$, where Q_a^λ is the answerable subquery of Q^λ.* □

Proof: (Sketch) *If*: If for each $\lambda \in \Omega(Q)$, $Q_a^\lambda \subseteq Q$. As shown in Figure 2, $Q = \bigcup_{\lambda \in \Omega(Q)} Q^\lambda$. In addition, for each $\lambda \in \Omega(Q)$, $Q^\lambda \subseteq Q_a^\lambda$. Then $Q = \bigcup_{\lambda \in \Omega(Q)} Q^\lambda \subseteq \bigcup_{\lambda \in \Omega(Q)} Q_a^\lambda \subseteq Q$. Thus $Q = \bigcup_{\lambda \in \Omega(Q)} Q_a^\lambda$. Note all the ordinary subgoals in each Q_a^λ are answerable. *Only If*: If there is a total order in $\Omega(Q)$, say λ_1, such that $Q_a^{\lambda_1} \not\subseteq Q$. By Theorem 5.1 in [9], we can 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. ■

Theorem 5.2 gives an algorithm *CQACstable* (shown in Figure 3) that tests stability of CQAC's. For a stable CQAC Q on a database D, we can compute $ANS(Q, D)$ by computing $ANS(Q_a^\lambda, D)$ for each total order λ in $\Omega(Q)$, and taking the union of these answers.

EXAMPLE 5.3 Consider the query P in Example 5.1. It has the following 8 total orders:

[2] 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 $X_i < X_j$.

λ_1: $X < A = Y < B$; λ_2: $X < A < Y < B$; λ_3: $X < A < Y = B$; λ_4: $X < A < B < Y$;
λ_5: $X = A = Y < B$; λ_6: $X = A < Y < B$; λ_7: $X = A < Y = B$; λ_8: $X = A < B < Y$.

For each total order λ_i, we can write its corresponding query P^{λ_i}. For instance:

$$P^{\lambda_1} : ans(Y) \text{ :- } p(X), r(X,Y), r(Y,B), X < Y, Y < B$$

We then compute the answerable subquery $P_a^{\lambda_i}$. All the 8 answerable subqueries are contained in P. Therefore, query P is stable. Actually, the union of all the answerable subqueries except $P_a^{\lambda_5}$ is

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

whose answerable subquery is the query T_1 in Example 5.1. In addition, $P_a^{\lambda_5}$ is equivalent to the query T_2. Since P is equivalent to the union of the 8 answerable subqueries, we have $P = T_1 \cup T_2$. □

6 Stability of datalog queries

In this section we prove stability of datalog programs is undecidable. We then give a sufficient condition for stability of datalog programs. That is, if a datalog query has a feasible rule/goal graph with respect to its query goal, then the query is stable.

6.1 Undecidability

Theorem 6.1 *Stability of datalog programs is undecidable.* □

Proof: (Sketch) 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 theorem.[3] 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. Z is a new argument that does not appear in P_1 and P_2. We can show that $P_1 \subseteq P_2$ if and only if query $\mathcal{Q}$ is stable. ∎

6.2 A sufficient condition for stability of datalog programs

Consider the following example. Assume $flight$ is a finite relation with a binding pattern 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 two rules:

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

[3] The idea of the proof is borrowed from [6], Chapter 2.3.

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 reachable from the airport sfo, while query P_2 asks for airports from which the airport sfo is reachable. Query P_1 is stable, since 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. We can retrieve the airport a following this chain. Query P_2 is not stable, since there can always be an airport a' from which sfo is reachable, but a' cannot be retrieved from $flight$.

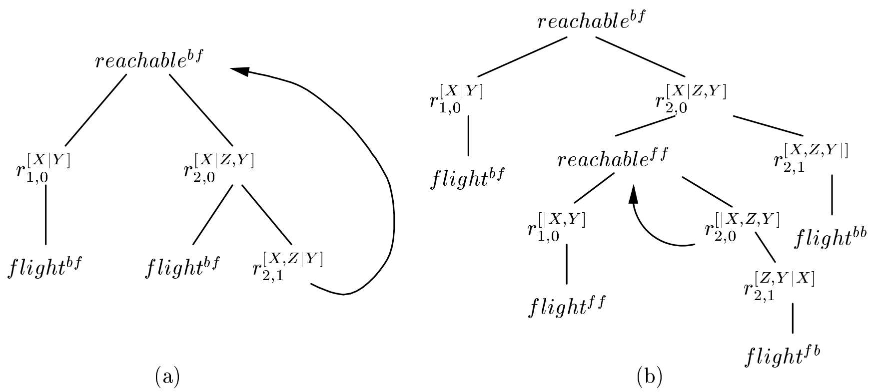

Figure 4: Two RGG's for the query goal $reachable^{bf}$

Query P_1 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 4(a) shows a rule/goal graph (RGG for short) [30] of the goal. It 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 4(b). Given a set of rules on EDB relations with binding restrictions, an RGG of a goal node is *feasible* if all its EDB goals in the RGG use only the adornments that are permitted by the EDB relations. For instance, Figure 4(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 4(b) is not feasible, since it has two EDB goals ($flight^{ff}$ and $flight^{fb}$) that are not permitted by the $flight$ relation.

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 [30] 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.

Theorem 6.2 *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: (Sketch) We apply the magic-sets transformation (as described in Chapter 13 in [30]) on the rules and the goal, and get a new set of rules $\mathcal{R}$ such that the relation for the query goal is the answer to the query. For any database D of the EDB relations, we can compute the complete answer to the query using a bottom-up evaluation of the rules $\mathcal{R}$. By the construction of the rules $\mathcal{R}$, each time we solve an EDB subgoal, we have enough bindings to evaluate this subgoal. ■

[26] gave an algorithm for testing the existence of a feasible RGG given a set of rules and a query goal. In some cases, even though a set of rules does not have a feasible RGG with respect to a query goal, the query may still be stable, since we may rewrite the rules to obtain a new set of rules that has a feasible RGG with respect to the query goal. For instance, 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 $flight(W, Z)$ cannot be bound. However, the new rules are equivalent to the old ones, and query P_1 on the new rules is still stable. This example shows a similar phenomenon as CQ's; that is, we need to "minimize" rules before checking the existence of a feasible RGG. However, minimizing datalog rules is much harder than minimizing CQ's and UCQ's. Shmueli [29] 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.

7 Related work

Several works considered binding restrictions in the context of answering queries using views [7, 17]. Rajaraman, Sagiv, and Ullman [27] 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 [7] solved the same problem by translating source restrictions into recursive datalog rules 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 [4]. [21] studied the problem of generating an executable plan based on source restrictions. [8, 32] studied query optimization in the presence of binding restrictions. [31] considered how to compute mediator restrictions given source restrictions. These four studies did not minimize a CQ before checking its feasibility. Thus, they regard the query Q_2 in Example 1.2 as an unsolvable query. [20] studied how to compute the maximal answer to a CQ in the presence of binding restrictions by borrowing bindings from relations not in the query. 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 CQ. [19, 24] study how to test query containment relative to views with binding restrictions.

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

References

[1] C. Beeri and R. Ramakrishnan. On the power of magic. In *Proc. of ACM Symposium on Principles of Database Systems (PODS)*, pages 269–283, 1987.

[2] T. Catarci. Web-based information access. *IFCIS International Conference on Cooperative Information Systems (CoopIS)*, pages 10–19, 1999.

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

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

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

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

[7] O. M. Duschka and A. Y. Levy. Recursive plans for information gathering. *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, IJCAI-97*, 1997.

[8] D. Florescu, A. Levy, I. Manolescu, and D. Suciu. Query optimization in the presence of limited access patterns. In *Proc. of ACM SIGMOD*, pages 311–322, 1999.

[9] A. Gupta, Y. Sagiv, J. D. Ullman, and J. Widom. Constraint checking with partial information. In *Proc. of ACM Symposium on Principles of Database Systems (PODS)*, pages 45–55. ACM Press, 1994.

[10] L. M. Haas, D. Kossmann, E. L. Wimmers, and J. Yang. Optimizing queries across diverse data sources. In *Proc. of VLDB*, pages 276–285, 1997.

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

[12] Z. Ives, D. Florescu, M. Friedman, A. Levy, and D. Weld. An adaptive query execution engine for data integration. In *Proc. of ACM SIGMOD*, pages 299–310, 1999.

[13] V. Josifovski and T. Risch. Integrating heterogenous overlapping databases through object-oriented transformations. In *Proc. of VLDB*, pages 435–446, 1999.

[14] Z. Kedad and M. Bouzeghoub. Discovering view expressions from a multi-source information system. *IFCIS International Conference on Cooperative Information Systems (CoopIS)*, pages 57–68, 1999.

[15] S. Kerr, A. Gal, and J. Mylopoulos. Information services for the web: Building and maintaining domain models. *IFCIS International Conference on Cooperative Information Systems (CoopIS)*, pages 4–13, 1998.

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

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

[18] C. Li. Computing complete answers to queries in the presence of limited access patterns (extended version). *Technical report, Computer Science Dept., Stanford Univ.*, `http://www-db.stanford.edu/~chenli/pubs/compbp-long.ps`, 1999.

[19] C. Li. Testing query containment in the presence of limited access patterns. *Technical report, Computer Science Dept., Stanford Univ.*, 1999.

[20] C. Li and E. Chang. Query planning with limited source capabilities. *International Conference on Data Engineering (ICDE)*, pages 401–412, 2000.

[21] C. Li, R. Yerneni, V. Vassalos, H. Garcia-Molina, Y. Papakonstantinou, J. D. Ullman, and M. Valiveti. Capability based mediation in TSIMMIS. In *Proc. of ACM SIGMOD*, pages 564–566, 1998.

[22] L. Liu, C. Pu, and W. Han. XWRAP: An XML-enabled wrapper construction system for web information sources. *International Conference on Data Engineering (ICDE)*, pages 611–621, 2000.

[23] R. J. Miller. Using schematically heterogeneous structures. In *Proc. of ACM SIGMOD*, pages 189–200, 1998.

[24] T. Millstein, A. Levy, and M. Friedman. Query containment for data integration systems. In *Proc. of ACM Symposium on Principles of Database Systems (PODS)*, 2000.

[25] T. Milo and S. Zohar. Using schema matching to simplify heterogeneous data translation. In *Proc. of VLDB*, pages 122–133, 1998.

[26] K. 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.

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

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

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

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

[31] R. Yerneni, C. Li, H. Garcia-Molina, and J. D. Ullman. Computing capabilities of mediators. In *Proc. of ACM SIGMOD*, pages 443–454, 1999.

[32] R. Yerneni, C. Li, J. D. Ullman, and H. Garcia-Molina. Optimizing large join queries in mediation systems. *International Conference on Database Theory (ICDT)*, pages 348–364, 1999.

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