Processing Unions of Conjunctive Queries with Negation under Limited Access Patterns

Alan Nash[1] and Bertram Ludäscher[2]

[1] Department of Mathematics, anash@math.ucsd.edu
[2] San Diego Supercomputer Center, ludaesch@sdsc.edu
University of California, San Diego

Abstract. We study the problem of answering queries over sources with limited access patterns. The problem is to decide whether a given query Q is *feasible*, i.e., equivalent to an *executable* query Q' that observes the limited access patterns given by the sources. We characterize the complexity of deciding feasibility for the classes $\mathrm{CQ}^\neg$ (conjunctive queries with negation) and $\mathrm{UCQ}^\neg$ (unions of $\mathrm{CQ}^\neg$ queries): Testing feasibility is just as hard as testing containment and therefore $\mathbf{\Pi}_2^{\mathrm{P}}$-complete. We also provide a uniform treatment for CQ, UCQ, $\mathrm{CQ}^\neg$, and $\mathrm{UCQ}^\neg$ by devising a single algorithm which is optimal for each of these classes. In addition, we show how one can often avoid the worst-case complexity by certain approximations: At compile-time, even if a query Q is not feasible, we can find efficiently the minimal executable query containing Q. For query answering at runtime, we devise an algorithm which may report complete answers even in the case of infeasible plans and which can indicate to the user the degree of completeness for certain incomplete answers.

1 Introduction

We study the problem of answering queries over sources with limited query capabilities. The problem arises naturally in the context of database integration and query optimization in the presence of limited source capabilities (e.g., see [PGH98,FLMS99]). In particular, for any database mediator system that supports not only conventional SQL databases, but also sources with *access pattern restrictions* [LC01,Li03], it is important to come up with query plans which observe those restrictions. Most notably, the latter occurs for sources which are modeled as *web services* [WSD03]. For the purposes of query planning, a web service operation can be seen as a remote procedure call, corresponding to a limited query capability which requires certain arguments of the query to be bound (the input arguments), while others may be free (the output arguments).

Web Services as Relations with Access Patterns. A *web service operation* can be seen as a function *op*: $x_1, \ldots, x_n \to y_1, \ldots, y_m$ having an *input message* (*request*) with n arguments (*parts*), and an *output message* (*response*) with m parts [WSD03, Part 2, Sec. 2.2]. For example, op_B: *author* $\to \{(\mathit{isbn}, \mathit{title})\}$ may implement a book search service, returning for a given author A a list of books

E. Bertino et al. (Eds.): EDBT 2004, LNCS 2992, pp. 422–440, 2004.

authored by A. We model such operations as *relations with access pattern*, here: $B^{\mathrm{oio}}(\mathit{isbn}, \mathit{author}, \mathit{title})$, where the access pattern 'oio' indicates that a value for the second attribute must be given as input, while the other attribute values can be retrieved as output. In this way, a family of web service operations over k attributes can be concisely described as a relation $R(a_1, \ldots, a_k)$ with an associated set of access patterns. Thus, queries become declarative specifications for web service composition.

An important problem of query planning over sources with access pattern restrictions is to determine whether a query Q is *feasible*, i.e., equivalent to an *executable query plan* Q' that observes the access patterns.

Example 1 The following conjunctive query[1] with negation

$$Q(i, a, t) \longleftarrow B(i, a, t), C(i, a), \neg L(i)$$

asks for books available through a store B which are contained in a catalog C, but not in the local library L. Let the only access patterns be B^{ioo}, B^{oio}, C^{oo}, and L^{o}. If we try to execute Q from left to right, neither pattern for B works since we either lack an ISBN i or an author a. However, Q is *feasible* since we can execute it by first calling $C(i, a)$ which binds both i and a. After that, calling $B^{\mathrm{ioo}}(i, a, t)$ or $B^{\mathrm{oio}}(i, a, t)$ will work, resulting in an executable plan. In contrast, calling $\neg L(i)$ first and then B does not work: a negated call can only filter out answers, but cannot produce any new variable bindings.

This example shows that for some queries which are not executable, simple reordering can yield an executable plan. However there are queries which cannot be reordered yet are feasible.[2] This raises the question of how to determine whether a query is feasible and how to obtain "good approximations" in case the query is *not* feasible. Clearly, these questions depend on the class of queries under consideration. For example, feasibility is undecidable for Datalog queries [LC01] and for first-order queries [NL04]. On the other hand, feasibility is decidable for subclasses such as conjunctive queries (CQ) and unions of conjunctive queries (UCQ) [LC01].

Contributions. We show that deciding feasibility for conjunctive queries with negation ($\mathrm{CQ}^\neg$) and unions of conjunctive queries with negation ($\mathrm{UCQ}^\neg$) is $\mathbf{\Pi}_2^{\mathrm{P}}$-complete, and present a corresponding algorithm, Feasible. Feasibility of CQ and UCQ was studied in [Li03]. We show that our uniform algorithm performs optimally on all these four query classes.

We also present a number of practical improvements and approximations for developers of database mediator systems: Plan* is an efficient polynomial-time algorithm for computing two plans Q^u and Q^o, which at runtime produce *underestimates* and *overestimates* of the answers to Q, respectively. Whenever Plan* outputs two identical Q^u and Q^o, we know at compile-time that Q is

[1] We write variables in lowercase.

[2] Li and Chang call this notion *stable* [LC01,Li03].

feasible without actually incurring the cost of the $\mathbf{\Pi}_2^{\mathrm{P}}$-complete feasibility test. In addition, we present an efficient runtime algorithm $\textsc{Answer}^\star$ which, given a database instance D, computes underestimates $\textsc{Answer}(Q^u, D)$ and overestimates $\textsc{Answer}(Q^o, D)$ of the exact answer. If Q is not feasible, $\textsc{Answer}^\star$ may still compute a complete answer and signal the completeness of the answer to the user at runtime. In case the answer is incomplete (or not known to be complete), $\textsc{Answer}^\star$ can often give a lower bound on the relative completeness of the answer.

Outline. The paper is organized as follows: Section 2 contains the preliminaries. In Section 3 we introduce our basic notions such as executable, orderable, and feasible. In Section 4 we present our main algorithms for computing execution plans, determining the feasibility of a query, and runtime processing of answers. In Section 5 we present the main theoretical results, in particular a characterization of the complexity of deciding feasibility of $\mathrm{UCQ}^\neg$ queries. Also we show how related algorithms can be obtained as special cases of our uniform approach. We summarize and conclude in Section 6.

2 Preliminaries

A *term* is a variable or constant. We use lowercase letters to denote terms. By $\bar{x}$ we denote a finite sequence of terms $x_1, \ldots, x_k$. A *literal* $\hat{R}(\bar{x})$ is an atom $R(\bar{x})$ or its negation $\neg R(\bar{x})$.

A *conjunctive query* Q is a formula of the form $\exists \bar{y}\ R_1(\bar{x}_1) \wedge \ldots \wedge R_n(\bar{x}_n)$. It can be written as a Datalog rule $Q(\bar{z}) \longleftarrow R_1(\bar{x}_1), \ldots, R_n(\bar{x}_n)$. Here, the existentially-quantified variables $\bar{y}$ are among the $\bar{x}_i$ and the distinguished (answer) variables $\bar{z}$ in the head of Q are the remaining *free variables* of Q, denoted free(Q). Let vars(Q) denote all variables of Q; then we have free$(Q) =$ vars$(Q) \setminus \{\bar{y}\} = \{\bar{z}\}$. Conjunctive queries (CQ) are also known as SPJ (select-project-join) queries.

A *union of conjunctive queries* (UCQ) is a query Q of the form $Q_1 \vee \ldots \vee Q_k$ where each $Q_i \in$ CQ. If free$(Q) = \{\bar{z}\}$, then Q in rule form consists of k rules, one for each Q_i, all with the same head $Q(\bar{z})$.

A *conjunctive query with negation* ($\mathrm{CQ}^\neg$) is defined like a conjunctive query, but with literals $\hat{R}_i(\bar{x}_i)$ instead of atoms $R_i(\bar{x}_i)$. Hence a $\mathrm{CQ}^\neg$ query is an existentially quantified conjunction of positive or negated atoms.

A *union of conjunctive queries with negation* ($\mathrm{UCQ}^\neg$) is a query $Q_1 \vee \ldots \vee Q_k$ where each $Q_i \in \mathrm{CQ}^\neg$; the rule form consists of k $\mathrm{CQ}^\neg$-rules having the same head $Q(\bar{z})$.

For $Q \in \mathrm{CQ}^\neg$, we denote by Q^+ the conjunction of the positive literals in Q in the same order as they appear in Q and by Q^- the conjunction of the negative literals in Q in the same order as they appear in Q.

A CQ or $\mathrm{CQ}^\neg$ query is *safe* if every variable of the query appears in a positive literal in the body. A UCQ or $\mathrm{UCQ}^\neg$ query is safe if each of its CQ or $\mathrm{CQ}^\neg$ parts is safe and if all of them have the same free variables. In this paper we only consider safe queries.

3 Limited Access Patterns and Feasibility

Here we present the basic definitions for source queries with limited access patterns. In particular, we define the notions executable, orderable, and feasible. While the former two notions are syntactic in the sense that they can be decided by a simple inspection of a query, the latter notion is semantic, since feasibility is defined up to logic equivalence. An executable query can be seen as a *query plan*, prescribing how to execute the query. An orderable query can be seen as an "almost executable" plan (it just needs to be reordered to yield a plan). A feasible query, however, does not directly provide an execution plan. The problem we are interested in is how to determine whether such an executable plan exists and how to find it. These are two different, but related, problems.

Definition 1 (Access Pattern) An *access pattern* for a k-ary relation R is an expression of the form R^α where α is word of length k over the alphabet $\{\mathrm{i}, \mathrm{o}\}$.

We call the jth position of P an *input slot* if $\alpha(j) = \mathrm{i}$ and an *output slot* if $\alpha(j) = \mathrm{o}$.[3] At runtime, we *must* provide values for input slots, while for output slots such values are not required, i.e., "*bound is easier*" [Ull88].[4] In general, with access pattern R^α we may retrieve the set of tuples $\{\bar{y} \mid R(\bar{x}, \bar{y})\}$ as long as we supply the values of $\bar{x}$ corresponding to all input slots in R.

Example 2 (Access Patterns) Given the access patterns B^{ioo} and B^{oio} for the book relation in Example 1 we can obtain, e.g., the set $\{\langle a, t\rangle \mid B(i, a, t)\}$ of authors and titles given an ISBN i and the set $\{t \mid \exists i\, B(i, a, t)\}$ of titles given an author a, but we cannot obtain the set $\{\langle a, t\rangle \mid \exists i\, B(i, a, t)\}$ of authors and titles, given no input.

Definition 2 (Adornment) Given a set $\mathcal{P}$ of access patterns, a $\mathcal{P}$-*adornment* on $Q \in \mathrm{UCQ}^\neg$ is an assignment of access patterns from $\mathcal{P}$ to relations in Q.

Definition 3 (Executable) $Q \in \mathrm{CQ}^\neg$ is $\mathcal{P}$-*executable* if $\mathcal{P}$-adornments can be added to Q so that every variable of Q appears first in an output slot of a non-negated literal. $Q \in \mathrm{UCQ}^\neg$ with $Q := Q_1 \vee \ldots \vee Q_k$ is $\mathcal{P}$-*executable* if every Q_i is $\mathcal{P}$-executable.

We consider the query which returns no tuples, which we write `false`, to be (vacuously) executable. In contrast, we consider the query with an empty body, which we write `true`, to be non-executable. We may have both kinds of queries in ans(Q) defined below. From the definitions, it follows that executable queries are safe. The converse is false.

An executable query provides a query *plan*: execute each rule separately (possibly in parallel) from left to right.

[3] Other authors use 'b' and 'f' for bound and free, but we prefer to reserve these notions for variables under or not under the scope of a quantifier, respectively.

[4] If a source does *not* accept a value, e.g., for y in $R^{\mathrm{io}}(x, y)$, one can ignore the y binding and call $R(x, y')$ with y' unbound, and afterwards execute the join for $y' = y$.

Definition 4 (Orderable) $Q \in \mathrm{UCQ}^{\neg}$ with $Q := Q_1 \vee \ldots \vee Q_k$ is $\mathcal{P}$-*orderable* if for every $Q_i \in \mathrm{CQ}^{\neg}$ there is a permutation Q'_i of the literals in Q_i so that $Q' := Q'_1 \vee \ldots \vee Q'_k$ is $\mathcal{P}$-executable.

Clearly, if Q is executable, then Q is orderable, but not conversely.

Definition 5 (Feasible) $Q \in \mathrm{UCQ}^{\neg}$ is $\mathcal{P}$-*feasible* if it is equivalent to a $\mathcal{P}$-executable $Q' \in \mathrm{UCQ}^{\neg}$.

Clearly, if Q is orderable, then Q is feasible, but not conversely.

Example 3 (Feasible, Not Orderable) Given access patterns B^{ioo}, B^{oio}, L^{o},

$$Q(a) \longleftarrow B(i,a,t), L(i), B(i',a',t)$$
$$Q(a) \longleftarrow B(i,a,t), L(i), \neg B(i',a',t)$$

is not orderable since i' and a' cannot be bound, but feasible because this query is equivalent to the executable query $Q'(a) \longleftarrow L(i), B(i,a,t)$.

Usually, we have in mind a fixed set $\mathcal{P}$ of access patterns and then we simply say executable, orderable, and feasible instead of $\mathcal{P}$-executable, $\mathcal{P}$-orderable, and $\mathcal{P}$-feasible. The following two definitions and the algorithm in Figure 1 are small modifications of those presented in [LC01].

Definition 6 (Answerable Literal) Given $Q \in \mathrm{CQ}^{\neg}$, we say that a literal $\hat{R}(\bar{x})$ (not necessarily in Q) is Q-*answerable* if there is an executable $Q^R \in \mathrm{CQ}^{\neg}$ consisting of $\hat{R}(\bar{x})$ and literals in Q.

Definition 7 (Answerable Part ans(Q)) If $Q \in \mathrm{CQ}^{\neg}$ is unsatisfiable then $\mathrm{ans}(Q) = \texttt{false}$. If Q is satisfiable, $\mathrm{ans}(Q)$ is the query given by the Q-answerable literals in Q, in the order given by the algorithm Answerable (see Figure 1). If $Q \in \mathrm{UCQ}^{\neg}$ with $Q = Q_1 \vee \ldots \vee Q_k$ then $\mathrm{ans}(Q) = \mathrm{ans}(Q_1) \vee \ldots \vee \mathrm{ans}(Q_k)$.

Notice that the answerable part $\mathrm{ans}(Q)$ of Q is executable whenever it is safe.

Proposition 1 *$Q \in \mathrm{CQ}^{\neg}$ is orderable iff every literal in Q is Q-answerable.*

Proposition 2 *There is a quadratic-time algorithm for computing* $\mathrm{ans}(Q)$.

The algorithm is given in Figure 1.

Corollary 3 *There is a quadratic-time algorithm for checking whether $Q \in \mathrm{UCQ}^{\neg}$ is orderable.*

In Section 5.1 we define and discuss containment of queries and in Section 5.2 we prove the following proposition. Query P is said to be *contained* in query Q (in symbols, $P \sqsubseteq Q$) if for every instance D, $\textsc{Answer}(P,D) \subseteq \textsc{Answer}(Q,D)$.

Proposition 4 *If $Q \in \mathrm{UCQ}^{\neg}$, then $Q \sqsubseteq \mathrm{ans}(Q)$.*

Corollary 5 *If $Q \in \mathrm{UCQ}^{\neg}$, $\mathrm{ans}(Q)$ is safe, and $\mathrm{ans}(Q) \sqsubseteq Q$, then Q is feasible.*

Proof If $\mathrm{ans}(Q) \sqsubseteq Q$ then $\mathrm{ans}(Q) \equiv Q$ and therefore, since $\mathrm{ans}(Q)$ is safe and thus executable, Q is feasible.

We show in Section 5 that the converse also holds; this is one of our main results.

```
Input:  – CQ¬ query Q = L1 ∧ ... ∧ Lk over a schema with access patterns P
Output: – ans(Q), the answerable part A of Q

procedure ANSWERABLE(Q, P)
  if UNSATISFIABLE(Q) then return false
  A := ∅;   B := ∅  /* initialize answerable literals and bound variables */
  repeat
    done := true
    for i := 1 to k do
      if Li ∉ A and ( vars(Li) ⊆ B or (positive(Li) and invars(Li) ⊆ B) ) then
        A := A ∧ Li ;   B := B ∪ vars(Li);   done := false
  until done
  return A
```

Fig. 1. Algorithm ANSWERABLE for $\mathrm{CQ}^\neg$ queries

4 Computing Plans and Answering Queries

Given a $\mathrm{UCQ}^\neg$ query $Q = Q_1 \vee \cdots \vee Q_n$ over a relational schema with access pattern restrictions $\mathcal{P}$, our goal is to find executable plans for Q which satisfy $\mathcal{P}$. As we shall see such plans may not always exist and deciding whether Q is feasible, i.e., equivalent to some executable Q' is a hard problem ($\mathbf{\Pi}_2^{\mathrm{P}}$-complete). On the other hand, we will be able to obtain efficient approximations, both at compile-time and at runtime. By *compile-time* we mean the time during which the query is being processed, before any specific database instance D is considered or available. By *runtime* we mean the time during which the query is executed against a specific database instance D. For example, feasibility is a compile-time notion, while completeness (of an answer) is a runtime notion.

4.1 Compile-Time Processing

Let us first consider the case of an individual $\mathrm{CQ}^\neg$ query $Q = L_1 \wedge \ldots \wedge L_k$ where each L_i is a literal. Figure 1 depicts a simple and efficient algorithm ANSWERABLE to compute $\mathrm{ans}(Q)$, the answerable part of Q: First we handle the special case that Q is unsatisfiable. In this case we return `false`. Otherwise, at every stage, we will have a set of input variables (i.e., variables with bindings) B and an executable sub-plan A. Initially, A and B are empty. Now we iterate, each time looking for at least one more answerable literal L_i that can be handled with the bindings B we have so far ($\mathrm{invars}(L_i)$ gives the variables in L_i which are in input slots). If we find such answerable literal L_i, we add it to A and we update our variable bindings B. When no such L_i is found, we exit the outer loop. Obviously, ANSWERABLE is polynomial (quadratic) time in the size of Q.

We are now ready to consider the general case of computing execution plans for a $\mathrm{UCQ}^\neg$ query Q (Figure 2). For each $\mathrm{CQ}^\neg$ query Q_i of Q, we compute its answerable part $A_i := \mathrm{ans}(Q_i)$ and its unanswerable part $U_i := Q_i \setminus A_i$. As the underestimate of Q_i^u, we consider A_i if U_i is empty; else we dismiss Q_i

Input: – UCQ$^\neg$ query $Q(\bar{x}) = Q_1 \vee \cdots \vee Q_n$ over a schema with access patterns $\mathcal{P}$
Output: – execution plans Q^u (underestimates), Q^o (overestimates)

procedure $\text{PLAN}^\star(Q)$
  for $i := 1$ to n do
    $A_i := \text{ANSWERABLE}(Q_i, \mathcal{P})$; $U_i := Q_i \setminus A_1$
    if $U_i = \emptyset$ then $Q_i^u := A_i$ else $Q_i^u :=$ `false`
    $\bar{v} := \bar{x} \setminus \text{vars}(A_i)$
    $Q_i^o := A_i \wedge (\, \bar{v} = \overline{\texttt{null}}\,)$
  $Q^u := Q_1^u \vee \cdots \vee Q_n^u$; $Q^o := Q_1^o \vee \cdots \vee Q_n^o$
  output Q^u, Q^o

Fig. 2. Algorithm $\text{PLAN}^\star$ for UCQ$^\neg$ queries

altogether for the underestimate. Either way, we ensure that $Q_i^u \sqsubseteq Q_i$. For the overestimate Q_i^o we give U_i the "benefit of doubt" and consider that it could be true. However, we need to consider the case that not all variables $\bar{x}$ in the head of the query occur in the answerable part A_i: some may appear only in U_i, so we cannot return a value for them. Hence we set the variables in $\bar{x}$ which are not in A_i to `null`. This way we ensure that $Q_i \sqsubseteq Q_i^o$, except when Q_i^o has null values. We have to interpret tuples with nulls carefully (see Section 4.2). Clearly, if all U_i are empty, then $Q^u = Q^o$ and all Q_i can be executed in the order given by ANSWERABLE, so Q is orderable and thus feasible. Also note that $\text{PLAN}^\star$ is efficient, requiring at most quadratic time.

Example 4 (Underestimate, Overestimate Plans) Consider the following query $Q = Q_1 \vee Q_2$ with the access patterns $\mathcal{P} = \{S^{\text{o}}, R^{\text{oo}}, B^{\text{ii}}, T^{\text{oo}}\}$.

$$Q_1(x,y) \longleftarrow \neg S(z), R(x,z), B(x,y)$$
$$Q_2(x,y) \longleftarrow T(x,y)$$

Although we can use $S(z)$ to produce bindings for z, this is not the case for its negation $\neg S(z)$. But by moving $R(x,z)$ to the front of the first disjunct, we can first bind z and then test against the filter $\neg S(z)$. However, we cannot satisfy the access pattern for B. Hence, we will end up with the following plans for $Q^u = Q_1^u \vee Q_2^u$ and $Q^o = Q_1^o \vee Q_2^o$.

$$Q_1^u(x,y) \longleftarrow \texttt{false}$$
$$Q_2^u(x,y) \longleftarrow T(x,y)$$
$$Q_1^o(x,y) \longleftarrow R(x,z), \neg S(z), y = \texttt{null}$$
$$Q_2^o(x,y) \longleftarrow T(x,y)$$

Note that the unanswerable part $U_1 = \{B(x,y)\}$ results in an underestimate Q_1^u equivalent to `false`, so Q_1^u can be dropped from Q^u (the unanswerable $B(x,y)$ is also responsible for the infeasibility of this plan). In the overestimate, $R(x,z)$ is moved in front of $\neg S(z)$, and $B(x,y)$ is replaced by a special condition equating the unknown value of y with `null`.

Input: – UCQ$^\neg$ query $Q(\bar{x}) = Q_1 \vee \cdots \vee Q_n$ over a schema with access patterns
Output: – true if Q is feasible, false otherwise

```
procedure FEASIBLE(Q)
  (Q^u, Q^o) := PLAN*(Q)
  if Q^u = Q^o then return true
  if Q^o contains null then return false else return Q^o ⊑ Q
```

Fig. 3. Algorithm FEASIBLE for UCQ$^\neg$ queries

Feasibility Test. While PLAN* is an efficient way to compute plans for a query Q, if it returns $Q^u \neq Q^o$ then we do not know whether Q is feasible. One way, discussed below, is to not perform any static analysis in addition to PLAN* and just "wait and see" what results Q^u and Q^o produce at runtime. This approach is particularly useful for ad-hoc, one-time queries.

On the other hand, when designing integrated views of a mediator system over distributed sources and web services, it is desirable to establish at view definition time that certain queries or views are feasible and have an equivalent executable plan for all database instances. For such "view design" and "view debugging" scenarios, a full static analysis using algorithm FEASIBLE in Figure 3 is desirable. First, FEASIBLE calls PLAN* to compute the two plans Q^u and Q^o. If Q^u and Q^o coincide, then Q is feasible. Similarly, if the overestimate contains some CQ$^\neg$ sub-query in which a `null` value occurs, we know that Q cannot be feasible (since then ans(Q) is unsafe). Otherwise, Q may still be feasible, i.e., if ans(Q) (= overestimate Q^o in this case) is contained in Q. The complexity of FEASIBLE is dominated by the $\mathbf{\Pi}_2^{\mathrm{P}}$-complete containment check $Q^o \sqsubseteq Q$.

4.2 Runtime Processing

The worst-case complexity of FEASIBLE seems to indicate that in practice and for large queries there is no hope to obtain plans having complete answers. Fortunately, the situation is not that bad after all. First, as indicated above, we may use the outcome of the efficient PLAN* algorithm to at least in some cases decide feasibility at compile-time (see first part of FEASIBLE up to the containment test). Perhaps even more important, from a practical point of view, is the ability to decide completeness of answers dynamically, i.e., at runtime.

Consider algorithm ANSWER* in Figure 4. We first let PLAN* compute the two plans Q^u and Q^o and evaluate them on the given database instance D to obtain the underestimate and overestimate ans_u and ans_o, respectively. If the difference Δ between them is empty, then we know the answer is complete even though the query may not be feasible. Intuitively, the reason is that an unanswerable part which causes the infeasibility may in fact be irrelevant for a specific query.

Input: – UCQ$^\neg$ query $Q(\bar{x}) = Q_1 \vee \cdots \vee Q_n$ over schema $\mathbf{R}$ with access patterns
– D a database instance over $\mathbf{R}$
Output: – underestimate ans_u
– difference Δ to overestimate ans_o
– completeness information

procedure $\text{ANSWER}^\star(Q)$
$(Q^u, Q^o) := \text{PLAN}^\star(Q)$
$ans_u := \text{ANSWER}(Q^u, D)$; $ans_o := \text{ANSWER}(Q^o, D)$; $\Delta := ans_o \setminus ans_u$
output ans_u
if $\Delta = \emptyset$ then output "answer is complete"
else
output "answer is not known to be complete"
output "these tuples may be part of the answer:" Δ
if Δ has no `null` values then
output "answer is at least" $\frac{|ans_u|}{|ans_o|}$ "complete"
/* *optional: minimize Δ using domain enumeration for U_i* */

Fig. 4. Algorithm $\text{ANSWER}^\star(\text{UCQ}^\neg)$ for runtime handling of plans

Example 5 (Not Feasible, Runtime Complete) Consider the plans created for the query in Example 4 (here we dropped the unsatisfiable Q_1^u):

$$\begin{aligned} Q_2^u(x,y) &\longleftarrow T(x,y) \\ Q_1^o(x,y) &\longleftarrow R(x,z), \neg S(z), y = \texttt{null} \\ Q_2^o(x,y) &\longleftarrow T(x,y) \end{aligned}$$

Given that B^{ii} is the only access pattern for B, the query Q_1 in Example 4 is not feasible since we cannot create y bindings for $B(x,y)$. However, for a given database instance D, it may happen that the answerable part $R(x,z), \neg S(z)$ does not produce any results. In that case, the unanswerable part $B(x,z)$ is irrelevant and the answer obtained is still complete.

Sometimes it is not accidental that certain disjuncts evaluate to false, but rather it follows from some underlying semantic constraints, in which case the omitted unanswerable parts do not compromise the completeness of the answer.

Example 6 (Dependencies) In the previous example, if $R.z$ is a foreign key referencing $S.z$, then always $\{\, z \mid R(x,z)\} \subseteq \{\, z \mid S(z)\}$. Therefore, the first disjunct $Q_1^o(x,y)$ can be discarded at compile-time by a semantic optimizer. However, even in the absence of such checks, our runtime processing can still recognize this situation and report a complete answer for this infeasible query.

In the BIRN mediator [GLM03], when unfolding queries against global-as-view defined integrated views into UCQ$^\neg$ plans, we have indeed experienced query plans with a number of unsatisfiable (with respect to some underlying, implicit integrity constraints) CQ$^\neg$ bodies. In such cases, when plans are redundant or partially unsatisfiable, our runtime handling of answers allows to report complete answers even in cases when the feasibility check fails or when the semantic optimization cannot eliminate the unanswerable part. In Figure 4, we know that ans_u is complete if Δ is empty, i.e., the overestimate plan Q^o has not contributed new answers. Otherwise we cannot know whether the answer is complete. However, if Δ does not contain `null` values, we can quantify the completeness of the underestimate relative to the overestimate.

We have to be careful when interpreting tuples with nulls in the overestimate.

Example 7 (Nulls) Let us now assume that $R(x, z), \neg S(z)$ from above holds for some variable binding. Such a binding, say $\beta = \{x/\mathtt{a}, z/\mathtt{b}\}$, gives rise to an overestimate tuple $Q_1^o(\mathtt{a}, \mathtt{null})$.

How should we interpret a tuple like $(\mathtt{a}, \mathtt{null}) \in \Delta$? The given variable binding $\beta = \{x/\mathtt{a}, z/\mathtt{b}\}$ gives rise to the following partially instantiated query:

$$Q_1^o(\mathtt{a}, y) \longleftarrow R(\mathtt{a}, \mathtt{b}), \neg S(\mathtt{b}), B(\mathtt{a}, y).$$

Given the access pattern B^{ii} we cannot know the contents of $\{y \mid B(\mathtt{a}, y)\}$. So our special `null` value in the answer means that there may be one or more y values such that $(\mathtt{a}, y)$ is in the answer to Q. On the other hand, there may be no such y in B which has `a` as its first component. So when $(\mathtt{a}, \mathtt{null})$ is in the answer, we can only infer that $R(\mathtt{a}, \mathtt{b})$ and $\neg S(\mathtt{b})$ are true for some value `b`; but we do not know whether indeed there is a matching $B(\mathtt{a}, y)$ tuple. The incomplete information on B due to the `null` value also explains why in this case we cannot give a numerical value for the completeness information in ANSWER*.

From Theorem 16 below it follows that the overestimates ans_o computed via Q^o cannot be improved, i.e., the construction is optimal. This is not the case for the underestimates as presented here.

Improving the Underestimate. The ANSWER* algorithm computes under- and overestimates ans_u, ans_o for UCQ$^\neg$ queries at runtime. If a query is feasible, then we will always have $ans_u = ans_o$, which is detected by ANSWER*. However, in the case of infeasible queries, there are still additional improvements that can be made. Consider the algorithm PLAN* in Figure 2: it divides a CQ$^\neg$ query Q_i into two parts, the answerable part A_i and the unanswerable part U_i. For each variable x_j which requires input bindings in U_i not provided by U_i, we can create a *domain enumeration view* $\mathsf{dom}(x_j)$ over the relations of the given schema and provide the bindings obtained in this way as partial domain enumerations to U_i.

Example 8 (Domain Enumeration) For our running example from above, instead of Q_1^u being `false`, we obtain an improved underestimate as follows:

$$Q_1^u(x, y) \longleftarrow R(x, z), \neg S(z), \mathsf{dom}(y), B(x, y)$$

where $\mathsf{dom}(y)$ could be defined, e.g., as the union of the projections of various columns from other relations for which we have access patterns with output slots: $\mathsf{dom}(x) \longleftarrow R(x, y) \vee R(z, x) \vee \ldots$

This domain enumeration approach has been used in various forms [DL97]. Note that in our setting of ANSWER* we can create a very dynamic handling of answers: if ANSWER* determines that $\Delta \neq \emptyset$, the user may want to decide at that point whether he or she is satisfied with the answer or whether the possibly costly domain enumeration views should be used. Similarly, the relative answer completeness provided by ANSWER* can be used to guide the user and/or the system when introducing domain enumeration views.

5 Feasibility of Unions of Conjunctive Queries with Negation

We now establish the complexity of deciding feasibility for safe UCQ$^\neg$ queries.

5.1 Query Containment

We need to consider query containment for UCQ$^\neg$ queries. In general, query P is said to be *contained* in query Q (in symbols, $P \sqsubseteq Q$) if for all instances D, ANSWER$(P, D) \subseteq$ ANSWER(Q, D). We write CONT$(\mathcal{L})$ for the following decision problem: For a class of queries $\mathcal{L}$, given $P, Q \in \mathcal{L}$ determine whether $P \sqsubseteq Q$.

For $P, Q \in$ CQ, a function σ: vars$(Q) \to$ vars(P) is a *containment mapping* if P and Q have the same free (distinguished) variables, σ is the identity on the free variables of Q, and, for every literal $R(\bar{y})$ in Q, there is a literal $R(\sigma\bar{y})$ in P.

Some early results in database theory are:

Proposition 6 *[CM77]* CONT(CQ) *and* CONT(UCQ) *are* **NP**-*complete.*

Proposition 7 *[SY80,LS93]* CONT(CQ$^\neg$) *and* CONT(UCQ$^\neg$) *are* $\mathbf{\Pi}_2^{\mathrm{P}}$-*complete.*

For many important special cases, testing containment can be done efficiently. In particular, the algorithm given in [WL03] for containment of safe CQ$^\neg$ and UCQ$^\neg$ uses an algorithm for CONT(CQ) as a subroutine. Chekuri and Rajaraman [CR97] show that containment of acyclic CQs can be solved in polynomial time (they also consider wider classes of CQs) and Saraiya [Sar91] shows that containment of CQs, in the case where no relation appears more than twice in the body, can be solved in linear time. By the nature of the algorithm in [WL03], these gains in efficiency will be passed on directly to the test for containment of CQs and UCQs (so the check will be in **NP**) and will also improve the test for containment of CQ$^\neg$ and UCQ$^\neg$.

5.2 Feasibility

Definition 8 (Feasibility Problem) FEASIBLE$(\mathcal{L})$ is the following decision problem: given $Q \in \mathcal{L}$ decide whether Q is feasible for the given access patterns.

Before proving our main results, Theorems 16 and 18, we need to establish a number of auxiliary results. Recall that we assume queries to be safe; in particular Theorems 12 and 13 hold only for safe queries.

Proposition 8 *$Q \in \mathrm{CQ}^{\neg}$ is unsatisfiable iff there exists a relation R and terms $\bar{x}$ so that both $R(\bar{x})$ and $\neg R(\bar{x})$ appear in Q.*

Proof Clearly if there are such R and $\bar{x}$ then Q is unsatisfiable. If not, then consider the frozen query $[Q^+]$ ($[Q^+]$ is a Herbrand model of Q^+). Clearly $[Q^+] \models Q$ so Q is satisfiable.

Therefore, we can check whether $Q \in \mathrm{CQ}^{\neg}$ is satisfiable in quadratic time: for every $R(\bar{x})$ in Q^+, look for $\neg R(\bar{x})$ in Q^-.

Proposition 9 *If $\hat{R}(\bar{x})$ is Q-answerable, then it is Q^+-answerable.*

Proposition 10 *If $Q \in \mathrm{CQ}^{\neg}$, $\hat{S}(\bar{x})$ is Q-answerable, and for every literal $R(\bar{x})$ in Q^+, $\neg R(\bar{x})$ is P-answerable, then $\hat{S}(\bar{x})$ is P-answerable.*

Proof If $\hat{S}(\bar{x})$ is Q-answerable, it is Q^+-answerable by Proposition 9. By definition, there must be executable Q' consisting of $\hat{S}(\bar{x})$ and literals from Q^+. Since every literal $R(\bar{x})$ in Q^+ is P-answerable, there must be executable P^R consisting of $R(\bar{x})$ and literals from P. Then the conjunction of all P^Rs is executable and consists of $\hat{S}(\bar{x})$ and literals from P. That is, $\hat{S}(\bar{x})$ is P-answerable.

Proposition 11 *If $P, Q \in \mathrm{CQ}$, σ: $\mathrm{vars}(Q) \to \mathrm{vars}(P)$ is a containment mapping (so $P \sqsubseteq Q$), and $\hat{R}(\sigma\bar{x})$ is Q-answerable, then $\hat{R}(\bar{x})$ is P-answerable.*

Proof If the hypotheses hold, there must be executable Q' consisting of $\hat{R}(\sigma\bar{x})$ and literals from Q. Then $P' = \sigma Q'$ consists of $\hat{R}(\bar{x})$ and literals from P. Since we can use the same adornments for P' as the ones we have for Q', P' is executable and therefore, $\hat{R}(\bar{x})$ is P-answerable.

Given $P, R \in \mathrm{CQ}^{\neg}$ where $P = (\exists\bar{x})\, P'$ and $Q = (\exists\bar{y})\, Q'$ and where P' and Q' are quantifier free (i.e., consist only of joins), we write P, Q to denote the query $(\exists\bar{x}, \bar{y})\, (P' \wedge Q')$. Recently, [WL03] gave the following theorems.

Theorem 12 *[WL03, Theorem 2] If $P, Q \in \mathrm{CQ}^{\neg}$ then $P \sqsubseteq Q$ iff P is unsatisfiable or there is a containment mapping σ: $\mathrm{vars}(Q) \to \mathrm{vars}(P)$ witnessing $P^+ \sqsubseteq Q^+$ such that, for every negative literal $\neg R(\bar{y})$ in Q, $R(\sigma\bar{y})$ is not in P and $P, R(\sigma\bar{y}) \sqsubseteq Q$.*

Theorem 13 *[WL03, Theorem 5] If $P \in \mathrm{CQ}^{\neg}$ and $Q \in \mathrm{UCQ}^{\neg}$ with $Q = Q_1 \vee \ldots \vee Q_k$ then $P \sqsubseteq Q$ iff P is unsatisfiable or if there is i ($1 \leqslant i \leqslant k$) and a containment mapping σ: $\mathrm{vars}(Q_i) \to \mathrm{vars}(P)$ witnessing $P^+ \sqsubseteq Q^+$ such that, for every negative literal $\neg R(\bar{y})$ in Q_i, $R(\sigma\bar{y})$ is not in P and $P, R(\sigma\bar{y}) \sqsubseteq Q$.*

Therefore, if $P \in \mathrm{CQ}^\neg$ and $Q \in \mathrm{UCQ}^\neg$ with $Q = Q_1 \vee \ldots \vee Q_k$, we have that $P \sqsubseteq Q$ iff there is a tree with root $P^+ \sqsubseteq Q_r^+$ for some r and where each node is of the form $P^+, N_1(\bar{x}_1), \ldots, N_m(\bar{x}_m) \sqsubseteq Q_s^+$ and represents a true containment except when $P, N_1(\bar{x}_1), \ldots, N_m(\bar{x}_m)$ is unsatisfiable, in which case also the node has no children. Otherwise, for some containment mapping

$$\sigma_s\colon \mathrm{vars}(Q_s^+) \to \mathrm{vars}(P^+, N_1(\bar{x}_1), \ldots, N_m(\bar{x}_m))$$

witnessing the containment, there is one child for every negative literal in Q_s. Each child is of the form $P^+, N_1(\bar{x}_1), \ldots, N_m(\bar{x}_m), N_{m+1}(\bar{x}_{m+1}) \sqsubseteq Q_t^+$ where $\bar{x}_{m+1} = \sigma_s(\bar{y})$ and $\neg N_{m+1}(\bar{y})$ appears in Q_s.

We will need the following two facts about this tree, in the special case where $Q \sqsubseteq E$ with E executable, in the proof of Theorem 16.

Lemma 14 *If* $\hat{R}(\bar{x})$ *is* $Q^+, N_1(\bar{x}_1), \ldots, N_m(\bar{x}_m)$*-answerable, it is* Q^+*-answerable.*

Proof By induction. It is obvious for $m = 0$. Assume that the lemma holds for m and that $\hat{R}(\bar{x})$ is $Q^+, N_1(\bar{x}_1), \ldots, N_{m+1}(\bar{x}_{m+1})$-answerable.

We have $Q^+, N_1(\bar{x}_1), \ldots, N_m(\bar{x}_m) \sqsubseteq E_s^+$ for some s witnessed by a containment mapping σ and $\bar{x}_{m+1} = \sigma(\bar{y})$ for some literal $\neg N_{m+1}(\bar{y})$ appearing in E_s. Since E_s is executable, by Propositions 1 and 9, $\neg N_{m+1}(\bar{y})$ is E_s^+-answerable. Therefore by Proposition 11, $\neg N_{m+1}(\bar{x})$ is $Q^+, N_1(\bar{x}_1), \ldots, N_m(\bar{x}_m)$-answerable and by the induction hypothesis, Q^+-answerable. Therefore, by Proposition 10 and the induction hypothesis, $\hat{R}(\bar{x})$ is Q^+-answerable.

Lemma 15 *If the conjunction* $Q, N_1(\bar{x}_1), \ldots, N_m(\bar{x}_m)$ *is unsatisfiable, then the conjunction* $\mathrm{ans}(Q), N_1(\bar{x}_1), \ldots, N_m(\bar{x}_m)$ *is also unsatisfiable.*

Proof If Q is satisfiable, but $Q, N_1(\bar{x}_1), \ldots, N_m(\bar{x}_m)$ is unsatisfiable, then by Proposition 8 we must have some $\neg N_i(\bar{x}_i)$ in Q. $N_i(\bar{x}_i)$ must have been added from some $N_i(\bar{y})$ in E_s and some containment map

$$\sigma_s\colon \mathrm{vars}(E_s^+) \to \mathrm{vars}(Q^+, N_1(\bar{x}_1), \ldots, N_{i-1}(\bar{x}_{i-1}))$$

satisfying $\sigma_s \bar{y} = \bar{x}$. Since E_s is executable, by Propositions 1 and 9, $\neg N_i(\bar{y})$ is E_s^+-answerable. Therefore by Proposition 11, $\neg N_i(\bar{x}_i)$ is $Q^+, N_1(\bar{x}_1), \ldots, N_m(\bar{x}_m)$-answerable and by Lemma 14, Q^+-answerable. Therefore, we must have $\neg N_i(\bar{x}_i)$ in $\mathrm{ans}(Q)$, so $\mathrm{ans}(Q), N_1(\bar{x}_1), \ldots, N_m(\bar{x}_m)$ is also unsatisfiable.

We include here the proof of Proposition 4 and then prove our main results, Theorems 16 and 18.

Proof (**Proposition 4**) For $Q \in \mathrm{CQ}$ this is clear since $\mathrm{ans}(Q)$ contains only literals from Q and therefore the identity map is a containment mapping from $\mathrm{ans}(Q)$ to Q. If $Q \in \mathrm{CQ}^\neg$ and Q is unsatisfiable, the result is obvious. Otherwise the identity is a containment mapping from $(\mathrm{ans}(Q))^+$ to Q^+. If a negative literal $\neg R(\bar{y})$ appears in $\mathrm{ans}(Q)$, then since $\neg R(\bar{y})$ also appears in Q, we have that $Q, R(\bar{y})$ is unsatisfiable, and therefore $Q \sqsubseteq \mathrm{ans}(Q)$ by Theorem 12.

Theorem 16 *If $Q \in \mathrm{UCQ}^\neg$, E is executable, and $Q \sqsubseteq E$, then $Q \sqsubseteq \mathrm{ans}(Q) \sqsubseteq E$. That is, $\mathrm{ans}(Q)$ is a minimal feasible query containing Q.*

Proof We have $Q \sqsubseteq \mathrm{ans}(Q)$ from Proposition 4. Set $A_i = \mathrm{ans}(Q_i)$. We know that for all i, $Q_i \sqsubseteq E$. We will show that $Q_i \sqsubseteq E$ implies $A_i \sqsubseteq E$, from which it follows that $\mathrm{ans}(Q) \sqsubseteq E$.

If Q_i is unsatisfiable, then A_i is also unsatisfiable, so $A_i \sqsubseteq E$ holds trivially, Therefore assume, to get a contradiction, that Q_i is satisfiable, $Q_i \sqsubseteq E$, and $A_i \not\sqsubseteq E$. Since Q_i is satisfiable and $Q_i \sqsubseteq E$, by [WL03, Theorem 4.3] we must have a tree with root $Q_i^+ \sqsubseteq E_r^+$ for some r and where each node is of the form $Q_i^+, N_1(\bar{x}_1), \ldots, N_m(\bar{x}_m) \sqsubseteq E_s^+$ and represents a true containment except when $Q_i, N_1(\bar{x}_1), \ldots, N_m(\bar{x}_m)$ is unsatisfiable, in which case also the node has no children. Otherwise, for some containment mapping

$$\sigma_s\colon \mathrm{vars}(E_s^+) \to \mathrm{vars}(Q_i^+, N_1(\bar{x}_1), \ldots, N_m(\bar{x}_m))$$

witnessing the containment there is one child for every negative literal in E_s. Each child is of the form $Q_i^+, N_1(\bar{x}_1), \ldots, N_m(\bar{x}_m), N_{m+1}(\bar{x}_{m+1}) \sqsubseteq E_t^+$ where $\bar{x}_{m+1} = \sigma_s(\bar{y})$ and $\neg N_{m+1}(\bar{y})$ appears in E_s.

Since $A_i \not\sqsubseteq E$, if in this tree we replace every Q_i^+ by A_i^+, by Lemma 15 we must have some non-terminal node where the containment doesn't hold. Accordingly, assume that $Q_i^+, N_1(\bar{x}_1), \ldots, N_m(\bar{x}_m) \sqsubseteq E_s^+$ and $A_i^+, N_1(\bar{x}_1), \ldots, N_m(\bar{x}_m) \not\sqsubseteq E_s^+$. For this to hold, there must be a containment mapping

$$\sigma_s\colon \mathrm{vars}(E_s^+) \to \mathrm{vars}(Q_i^+, N_1(\bar{x}_1), \ldots, N_m(\bar{x}_m))$$

which maps into some literal $R(\bar{x})$ which appears in Q_i^+ but not in A_i^+. That is, there must be some $\bar{y}$ so that $R(\bar{y})$ appears in E_s and $\sigma(\bar{y}) = \bar{x}$. By Propositions 1 and 9, since E_s is executable, $R(\bar{y})$ is E_s^+-answerable. By Proposition 11, $R(\bar{x})$ is $Q_i^+, N_1(\bar{x}_1), \ldots, N_m(\bar{x}_m)$-answerable and so, by Lemma 14, Q_i^+-answerable. Therefore, $R(\bar{x})$ is in A_i^+, which is a contradiction.

Corollary 17 *Q is feasible iff $\mathrm{ans}(Q) \sqsubseteq Q$.*

Theorem 18 $\mathrm{FEASIBLE}(\mathrm{UCQ}^\neg) \equiv_m^{\mathrm{P}} \mathrm{CONT}(\mathrm{UCQ}^\neg)$.

That is, determining whether a $\mathrm{UCQ}^\neg$ query is feasible is polynomial-time many-one equivalent to determining whether a $\mathrm{UCQ}^\neg$ query is contained in another $\mathrm{UCQ}^\neg$ query.

Proof One direction follows from Corollary 17 and Proposition 2. For the other direction, consider two queries $P, Q \in \mathrm{UCQ}^\neg$ where $P = P_1 \vee \ldots \vee P_k$. The query

$$P' := P_1, B(y) \vee \ldots \vee P_k, B(y)$$

where y is a variable not appearing in P or Q and B is a relation not appearing in P or Q with access pattern B^{i}. We give relations R appearing in P or Q output access patterns (i.e., $R^{\mathrm{ooo}\cdots}$). As a result, P and Q are both executable, but $P' \sqsubset P$ and P' is not feasible. We set $Q' := P' \vee Q$. Clearly, $\text{ans}(Q') \equiv P \vee Q$. If $P \sqsubseteq Q$, then $\text{ans}(Q') \equiv P \vee Q \equiv Q \sqsubseteq Q'$ so by Corollary 17, Q' is feasible. If $P \not\sqsubseteq Q$, then since $P' \sqsubset P$ and $P' \not\sqsubseteq Q$ we have $\text{ans}(Q') \equiv P \vee Q \not\sqsubseteq P' \vee Q \equiv Q'$ so again by Corollary 17, Q' is not feasible.

Since CONT(UCQ$^\neg$) is $\mathbf{\Pi}_2^{\mathrm{P}}$-complete, we have

Corollary 19 FEASIBLE(UCQ$^\neg$) *is* $\mathbf{\Pi}_2^{\mathrm{P}}$*-complete.*

UCQ$^\neg$ includes the classes CQ, UCQ, and CQ$^\neg$. We have the following strict inclusions $\text{CQ} \subsetneq \text{UCQ}, \text{CQ}^\neg \subsetneq \text{UCQ}^\neg$. Algorithm Feasible essentially consists of two steps: (i) compute $\text{ans}(Q)$, and (ii) test $\text{ans}(Q) \sqsubseteq Q$. Below we show that Feasible provides optimal processing for all the above subclasses of UCQ$^\neg$. Also, we compare Feasible to the algorithms given in [LC01].

5.3 Conjunctive Queries

Li and Chang [LC01] show that FEASIBLE(CQ) is **NP**-complete and provide two algorithms for testing feasibility of $Q \in \text{CQ}$:

- Find a minimal $M \in \text{CQ}$ so $M \equiv Q$, then check that $\text{ans}(M) = M$ (they call this algorithm CQstable).
- Compute $\text{ans}(Q)$, then check that $\text{ans}(Q) \sqsubseteq Q$ (they call this algorithm CQstable*).

The advantage of the latter approach is that $\text{ans}(Q)$ may be equal to Q, eliminating the need for the equivalence check. For conjunctive queries, algorithm Feasible is exactly the same as CQstable*.

Example 9 (CQ Processing) Consider access patterns F^{o} and B^{i} and the conjunctive query

$$Q(x) \longleftarrow F(x), B(x), B(y), F(z)$$

which is not orderable. Algorithm CQstable first finds the minimal $M \equiv Q$

$$M(x) \longleftarrow F(x), B(x)$$

then checks M for orderability (M is in fact executable). Algorithms CQstable* and Feasible first find $A := \text{ans}(Q)$

$$A(x) \longleftarrow F(x), B(x), F(z)$$

then check that $A \sqsubseteq Q$ holds (which is the case).

5.4 Conjunctive Queries with Union

Li and Chang [LC01] show that FEASIBLE(UCQ) is **NP**-complete and provide two algorithms for testing feasibility of $Q \in \text{UCQ}$ with $Q = Q_1 \vee \ldots \vee Q_k$:

– Find a minimal (with respect to union) $M \in$ UCQ so $M \equiv Q$ with $M = M_1 \vee \ldots \vee M_\ell$, then check that every M_i is feasible using either CQstable or CQstable* (they call this algorithm UCQstable)
– Take the union P of all the feasible Q_is, then check that $Q \sqsubseteq P$ (they call this algorithm UCQstable*). Clearly, $P \sqsubseteq Q$ holds by construction.

For UCQs, algorithm FEASIBLE is different from both of these and thus provides an alternate algorithm. The advantage of CQstable* and FEASIBLE over CQstable is that P or ans(Q) may be equal to Q, eliminating the need for the equivalence check.

Example 10 (UCQ Processing) Consider access patterns F^{o}, G^{o}, H^{o}, and B^{i} and the query

$$\begin{aligned} Q(x) &\longleftarrow F(x), G(x) \\ Q(x) &\longleftarrow F(x), H(x), B(y) \\ Q(x) &\longleftarrow F(x) \end{aligned}$$

Algorithm UCQstable first finds the minimal (with respect to union) $M \equiv Q$

$$M(x) \longleftarrow F(x)$$

then checks that M is feasible (it is). Algorithm UCQstable* first finds P, the union of the feasible rules in Q

$$\begin{aligned} P(x) &\longleftarrow F(x), G(x) \\ P(x) &\longleftarrow F(x) \end{aligned}$$

then checks that $Q \sqsubseteq P$ holds (it does). Algorithm FEASIBLE finds $A :=$ ans(Q) the union of the answerable part of each rule in Q

$$\begin{aligned} A(x) &\longleftarrow F(x), G(x) \\ A(x) &\longleftarrow F(x), H(x) \\ A(x) &\longleftarrow F(x) \end{aligned}$$

then checks that $A \sqsubseteq Q$ holds (it does).

5.5 Conjunctive Queries with Negation

Proposition 20 CONT(CQ$^\neg$) $\leqslant_m^{\mathrm{P}}$ FEASIBLE(CQ$^\neg$)

PROOF Assume $P, Q \in$ CQ$^\neg$ are given by

$$\begin{aligned} P(\bar{x}) &:= (\exists \bar{x}_0)(\hat{R}_1(\bar{x}_1) \wedge \ldots \wedge \hat{R}_k(\bar{x}_k)) \\ Q(\bar{x}) &:= (\exists \bar{y}_0)(\hat{S}_1(\bar{y}_1) \wedge \ldots \wedge \hat{S}_\ell(\bar{y}_\ell)) \end{aligned}$$

where the R_is and S_is are not necessarily distinct and the x_is and y_is are also not necessarily distinct. Then define

$$L(\bar{x}) := (\exists \bar{x}_0, \bar{y}_0, u, v)(T(u) \wedge \hat{R}'_1(u, \bar{x}_1) \wedge \ldots \wedge \hat{R}'_k(u, \bar{x}_k) \wedge \hat{S}'_1(v, \bar{y}_1) \wedge \ldots \wedge \hat{S}'_\ell(v, \bar{y}_\ell))$$

with access patterns T^{o}, $R_i'^{\mathrm{ioo}\cdots}$, $S_i'^{\mathrm{ioo}\cdots}$. Then clearly

$$\mathrm{ans}(L) = (\exists \bar{x}_0, u)(T(u) \wedge \hat{R}'_1(u, \bar{x}_1) \wedge \ldots \wedge \hat{R}'_k(u, \bar{x}_k))$$

and therefore $P \sqsubseteq Q$ iff $P \sqsubseteq P \wedge Q$ iff $\mathrm{ans}(L) \sqsubseteq L$ iff L is feasible. The second iff follows from the fact that every containment mapping η: $P \wedge Q \to P$ corresponds to a unique containment mapping η': $L \to \mathrm{ans}(L)$ and vice versa.

Since CONT(CQ$^\neg$) is $\mathbf{\Pi}_2^{\mathrm{P}}$-complete, we have

Corollary 21 FEASIBLE(CQ$^\neg$) *is* $\mathbf{\Pi}_2^{\mathrm{P}}$*-complete.*

6 Discussion and Conclusions

We have studied the problem of producing and processing executable query plans for sources with limited access patterns. In particular, we have extended the results by Li et al. [LC01,Li03] to conjunctive queries with negation (CQ$^\neg$) and unions of conjunctive queries with negation (UCQ$^\neg$). Our main theorem (Theorem 18) shows that checking feasibility for CQ$^\neg$ and UCQ$^\neg$ is equivalent to checking containment for CQ$^\neg$ and UCQ$^\neg$, respectively, and thus is $\mathbf{\Pi}_2^{\mathrm{P}}$-complete. Moreover, we have shown that our treatment for UCQ$^\neg$ nicely unifies previous results and techniques for CQ and UCQ respectively and also works optimally for CQ$^\neg$. In particular, we have presented a new uniform algorithm which is optimal for all four classes. We have also shown how we can often avoid the theoretical worst-case complexity, both by approximations at compile-time and by a novel runtime processing strategy. The basic idea is to avoid performing the computationally hard containment checks and instead (i) use two efficiently computable approximate plans Q^u and Q^o, which produce tight underestimates and overestimates of the actual query answer for Q (algorithm PLAN*), and defer the containment check in the algorithm FEASIBLE if possible, and (ii) use a runtime algorithm ANSWER*, which may report complete answers even in the case of infeasible plans, and which can sometimes quantify the degree of completeness. [Li03, Sec.7] employs a similar technique to the case of CQ. However, since union and negation are not handled, our notion of bounding the result from above and below is not applicable there (essentially, the underestimate is always empty when not considering union).

Although technical in nature, our work is driven by a number of practical engineering problems. In the Bioinformatics Research Network project [BIR03], we are developing a database mediator system for federating heterogeneous brain data [GLM03,LGM03]. The current prototype takes a query against a global-as-view definition and unfolds it into a UCQ$^\neg$ plan. We have used ANSWERABLE and a simplified version (without containment check) of PLAN* and ANSWER* in the system. Similarly, in the SEEK and SciDAC projects [SEE03,SDM03] we are building distributed scientific workflow systems which can be seen as procedural variants of the declarative query plans which a mediator is processing.

We are interested in extending our techniques to larger classes of queries and to consider the addition of integrity constraints. Even though many questions become undecidable when moving to full first-order or Datalog queries, we are interested in finding analogous compile-time and runtime approximations as presented in this paper.

Acknowledgements. Work supported by NSF-ACI 9619020 (NPACI), NIH 8P41 RR08605-08S1 (BIRN-CC), NSF-ITR 0225673 (GEON), NSF-ITR 0225676 (SEEK), and DOE DE-FC02-01ER25486 (SciDAC).

References

[BIR03] Biomedical Informatics Research Network Coordinating Center (BIRN-CC), University of California, San Diego. http://nbirn.net/, 2003.

[CM77] A. K. Chandra and P. M. Merlin. Optimal Implementation of Conjunctive Queries in Relational Data Bases. In *ACM Symposium on Theory of Computing (STOC)*, pp. 77–90, 1977.

[CR97] C. Chekuri and A. Rajaraman. Conjunctive query containment revisited. In *Intl. Conf. on Database Theory (ICDT)*, Delphi, Greece, 1997.

[DL97] O. M. Duschka and A. Y. Levy. Recursive plans for information gathering. In *Proc. IJCAI*, Nagoya, Japan, 1997.

[FLMS99] D. Florescu, A. Y. Levy, I. Manolescu, and D. Suciu. Query Optimization in the Presence of Limited Access Patterns. In *SIGMOD*, pp. 311–322, 1999.

[GLM03] A. Gupta, B. Ludäscher, and M. Martone. BIRN-M: A Semantic Mediator for Solving Real-World Neuroscience Problems. In *ACM Intl. Conference on Management of Data (SIGMOD)*, 2003. System demonstration.

[LC01] C. Li and E. Y. Chang. On Answering Queries in the Presence of Limited Access Patterns. In *Intl. Conference on Database Theory (ICDT)*, 2001.

[LGM03] B. Ludäscher, A. Gupta, and M. E. Martone. Bioinformatics: Managing Scientific Data. In T. Critchlow and Z. Lacroix, editors, *A Model-Based Mediator System for Scientific Data Management*. Morgan Kaufmann, 2003.

[Li03] C. Li. Computing Complete Answers to Queries in the Presence of Limited Access Patterns. *Journal of VLDB*, 12:211–227, 2003.

[LS93] A. Y. Levy and Y. Sagiv. Queries Independent of Updates. In *Proc. VLDB*, pp. 171–181, 1993.

[NL04] A. Nash and B. Ludäscher. Processing First-Order Queries under Limited Access Patterns. submitted for publication, 2004.

[PGH98] Y. Papakonstantinou, A. Gupta, and L. M. Haas. Capabilities-Based Query Rewriting in Mediator Systems. *Distributed and Parallel Databases*, 6(1):73–110, 1998.

[Sar91] Y. Saraiya. *Subtree elimination algorithms in deductive databases.* PhD thesis, Computer Science Dept., Stanford University, 1991.

[SDM03] Scientific Data Management Center (SDM). http://sdm.lbl.gov/sdmcenter/ and http://www.er.doe.gov/scidac/, 2003.

[SEE03] Science Environment for Ecological Knowledge (SEEK). http://seek.ecoinformatics.org/, 2003.

[SY80] Y. Sagiv and M. Yannakakis. Equivalences Among Relational Expressions with the Union and Difference Operators. *Journal of the ACM*, 27(4):633–655, 1980.

[Ull88] J. Ullman. The Complexity of Ordering Subgoals. In *ACM Symposium on Principles of Database Systems (PODS)*, 1988.

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

[WSD03] Web Services Description Language (WSDL) Version 1.2. http://www.w3.org/TR/wsdl12, June 2003.