# 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 finding executable query plans over distributed sources with limited access patterns. This problem is becoming increasingly important in the area of distributed query processing, most notably, web services. For the purposes of query planning, web services can be seen as remote procedure calls with input/output access pattern restrictions. The problem is to decide whether a given query $Q$ is *feasible*, i.e., whether one can find an equivalent *executable* query $Q'$ that observes the limited access patterns given by the sources. We characterize the complexity of deciding feasibility for the classes $\text{CQ}^\neg$ (conjunctive queries with negation) and $\text{UCQ}^\neg$ (unions of $\text{CQ}^\neg$ queries), which has been open until now: testing feasibility is just as hard as testing containment and therefore $\Pi_2^{\text{P}}$-complete. We also provide a *uniform treatment* for CQ, UCQ, $\text{CQ}^\neg$, and $\text{UCQ}^\neg$ by devising an 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 and at runtime. 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 finding executable query plans over distributed sources having 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 Service Operations.* We use the following abstraction of web services as relations with limited access patterns: A *web service interface* is a set of related *operations* $\{op_1, \ldots, op_k\}$, each of which has an associated *input message* (request) and *output message* (response).[1] The $n$ ($m$) *parts* of an input (output) message correspond to the $n$ input ($m$ output) parameters of the web service operation, respectively. For the purpose of distributed query planning, we can often consider a web service operation to be a function

$$op : x_1, \ldots, x_n \to \{\langle y_1, \ldots, y_m \rangle\}$$

mapping input $n$-tuples to a number of output $m$-tuples. We can model this operation as a simple relational view with *access pattern* (a.k.a. *binding pattern*) restrictions:

$$R_{op}^{\mathrm{i...i\,o...o}}(\bar{x}, \bar{y}) \longleftarrow \bar{y} \in op(\bar{x})$$

Here the access pattern 'i...i o...o' ($= \mathrm{i}^n\mathrm{o}^m$) of $R_{op}$ indicates that the first $n$ arguments $\bar{x} = x_1, \ldots, x_n$ serve as inputs and thus need to be bound, while the next $m$ arguments $\bar{y} = y_1, \ldots, y_m$ serve as outputs.

**Example 1 (Web Services)** Consider the following information integration example over a books schema with the usual attributes. Some data source may export the following operations as part of a web service:

$$op_1 : \mathrm{ISBN} \to \{\langle \mathrm{Authors}, \mathrm{Title}, \mathrm{Price} \rangle\}$$
$$op_2 : \mathrm{Title} \to \{\langle \mathrm{ISBN}, \mathrm{Authors}, \mathrm{Price} \rangle\}$$

The first operation returns, for a given ISBN, answer tuples containing the authors, title, and price, while the second operation returns, for a given title, a set of answer tuples having ISBN, author, and price information. We can consider these operations as simple *relational views with access patterns* over the *same* underlying book relation $B$:[2]

$$B^{\mathrm{iooo}}(x_i, x_a, x_t, x_p) \longleftarrow \langle x_a, x_t, x_p \rangle \in op_1(x_i)$$
$$B^{\mathrm{ooio}}(x_i, x_a, x_t, x_p) \longleftarrow \langle x_i, x_a, x_p \rangle \in op_2(x_t)$$

Here the access patterns 'iooo' and 'ooio' of $B$ indicate the input and output restrictions of each argument position of $B$ and thus correspond to the allowed access patterns for the two web service operations $op_1$ and $op_2$: we need at least the ISBN or a title to get any book information.

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

---

[1] This corresponds to the message pattern *in-out* [WSD03, Part 2, Section 2.2]

[2] We denote logic variables in lowercase $x, y, \ldots$ and relation symbols in uppercase $R, S, T, \ldots$

**Example 2 (Query Plans)** Consider the following conjunctive query with negation $Q$

$$Q(x_i, x_a, x_t, x_p) \longleftarrow B(x_i, x_a, x_t, x_p), C(x_i, \ldots), \neg L(x_t, \ldots)$$

asking for books available through $B$ which are contained in some catalog $C$, but not in the local library $L$. Assume that the access patterns are $B^{\mathrm{iooo}}$, $B^{\mathrm{ooio}}$, $C^{\mathrm{o}\cdots}$, and $L^{\mathrm{o}\cdots}$, respectively. The body of the query $Q$ is *not executable* as is (i.e., from left to right) because $B$ would be invoked first with no variable bound, violating both access patterns for $B$. However, the query becomes executable when we move the call to $C$ *before* the call to $B$. We say that the query $Q$ is *orderable* since one can rearrange it into an executable form. Had we moved $\neg L(x_t, \ldots)$ in front of $B$ instead of $C(x_i, \ldots)$, then the resulting query would not have been executable, in spite of $L$'s all-output access pattern. The reason is that a negated call can only be used to constrain or filter a set of answers, but cannot produce variable bindings.

The previous 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.[3] 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 of Datalog queries is undecidable [LC01] and a similar construction shows that it is also undecidable for first-order queries [NL03]. On the other hand, feasibility is decidable for subclasses such as conjunctive queries without (CQ) and with union (UCQ) [LC01].

*Contributions.* We present a new algorithm, FEASIBLE, for deciding the feasibility of conjunctive queries with negation ($\mathrm{CQ}^\neg$) and unions of conjunctive queries with negation ($\mathrm{UCQ}^\neg$), thereby extending previous results for CQ and UCQ [Li03]. Our construction also provides an elegant unifying treatment for all of CQ, UCQ, $\mathrm{CQ}^\neg$, and $\mathrm{UCQ}^\neg$; in particular we present a uniform algorithm that performs optimally for each of these classes. For our upper bounds, we use a recent elegant algorithm and result by Wei and Lausen [WL03] for containment of safe $\mathrm{UCQ}^\neg$ and UCQ. We provide matching lower bounds for FEASIBLE, thus characterizing the complexity of feasibility of $\mathrm{UCQ}^\neg$ and UCQ as $\Pi_2^{\mathrm{P}}$-complete. In addition to these new theoretical results on the complexity of deciding feasibility, we also present a number of practical improvements and approximations for developers of database mediator systems:[4] We present an efficient polynomial-time algorithm, PLAN$^\star$, which computes two plans $Q^u$ and $Q^o$, which at runtime produce *underestimates* and *overestimates* of the answers to $Q$, respectively. Whenever PLAN$^\star$ outputs two identical $Q^u$ and $Q^o$, we know at compile-time that $Q$ is feasible without actually incurring the cost

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

[4] The corresponding efficient algorithms are being added to a real-world database mediator system [BIR].

of the $\Pi_2^{\mathrm{P}}$-complete general feasibility test. In addition, we present an efficient runtime algorithm ANSWER$^\star$ which, given a database instance $D$, computes underestimates ANSWER$(Q^u, D)$ and overestimates ANSWER$(Q^o, D)$ of the exact answer. If $Q$ is not feasible, 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), ANSWER$^\star$ can often give a lower bound on the relative completeness of the answer.

*Outline.* The paper is organized as follows: In Section 2 we provide some preliminaries and terminology on the query classes being studied. In Section 3 we introduce basic notions such as executable, orderable, and feasible, which will be needed throughout the paper. 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 UCQ$^\neg$ queries. Also we show how related algorithms can be obtained as special cases of our uniform approach. Finally, we summarize our findings and discuss how our work relates to several real-world application projects in Section 6. We include some technical details and proofs in the appendix.

## 2 Preliminaries

We will need to talk about several kinds of queries, so we review their definitions here. We call a variable or a constant a *term*. Unless we explicitly disallow constants or it is clear from the context (for example under quantification), we use expressions of the form $\bar{x}$ to refer to a finite sequence of terms. We call an atomic predicate $R(\bar{x})$ or its negation $\neg R(\bar{x})$ a *literal*. We use $\hat{R}(\bar{x})$ to denote either $R(\bar{x})$ or $\neg R(\bar{x})$.

A *conjunctive query* (CQ) is a query $Q$ of the form

$$(\exists \bar{y})(R_1(\bar{x}_1) \wedge \ldots \wedge R_\ell(\bar{x}_\ell))$$

where each $R_i(\bar{x}_i)$ is an atomic predicate and where $\bar{y}$ is included in $\bar{x}_1, \ldots, \bar{x}_\ell$. That is, CQs are existential quantifications of conjunctions of positive literals or SPJ (select-project-join) queries. The variables $\bar{y}$ are *bound* or *non-distinguished*. The remaining variables are *free* or *distinguished* and we denote them by free$(Q)$. We denote all variables in $Q$ by vars$(Q)$ so we have free$(Q)$ := vars$(Q) \setminus \{\bar{y}\}$. Assume that in the case of $Q$ these variables are $\bar{z}$; then we can write $Q$ in rule form as follows

$$Q(\bar{z}) \longleftarrow R_1(\bar{x}_1), \ldots, R_\ell(\bar{x}_\ell)$$

(the existential quantification is implicit). We call $Q(\bar{z})$ the *head* and $R_1(\bar{x}_1), \ldots, R_\ell(\bar{x}_\ell)$ the *body* of the rule.

A *union of conjunctive queries* (UCQ) is a query $Q$ of the form

$$Q_1 \vee \ldots \vee Q_k$$

where each $Q_i \in \text{CQ}$. In other words, UCQs are unions of SPJ queries. If the free variables in $Q$ are $\bar{z}$, then to write $Q$ in rule form we simply give one rule for each $Q_i$, all with the same head $Q(\bar{z})$.

A *conjunctive query with negation* ($\text{CQ}^\neg$) is a query of the form

$$(\exists \bar{y})(\hat{R}_1(\bar{x}_1) \wedge \ldots \wedge \hat{R}_\ell(\bar{x}_\ell))$$

where each $\hat{R}_i(\bar{x}_i)$ is a literal and where $\bar{y}$ is included in $\bar{x}_1, \ldots, \bar{x}_\ell$. That is, $\text{CQ}^\neg$s are existential quantifications of conjunctions of literals (both positive and negative).

A *union of conjunctive queries with negation* ($\text{UCQ}^\neg$) is a query of the form

$$Q_1 \vee \ldots \vee Q_k$$

where each $Q_i \in \text{CQ}^\neg$. In other words, $\text{UCQ}^\neg$s are unions of $\text{CQ}^\neg$s.

For $Q \in \text{CQ}^\neg$, 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 $\text{CQ}^\neg$ query is *safe* if every variable of the query appears in a positive literal in the body. A UCQ or $\text{UCQ}^\neg$ query is safe if each of its CQ or $\text{CQ}^\neg$ 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

In this section we present the basic definitions for queries in the presence of 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 $R^\alpha$ where $\alpha$ is word of length $k$ over the alphabet $\{\text{i}, \text{o}\}$.

We call the $j$th position of $P$ an *input* slot if $\alpha(j) = \text{i}$ and an *output slot* if $\alpha(j) = \text{o}$.[5] At runtime, we *must* provide values for input slots in order to execute the query, while for output slots such values are not required (and instead can be provided by the source relation being queried).

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

In general, with access pattern $R^\alpha$ 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 $R$. We allow values to be supplied to output slots, but they are not required.

**Example 3 (Access Patterns)** Given the access patterns $B^{\text{iooo}}$ and $B^{\text{ooio}}$ on the book relation mentioned in the introduction, we can obtain the set $\{\langle x_a, x_p\rangle \mid (\exists x_i)B(x_i, x_a, x_t, x_p)\}$ of authors and prices given a title $x_t$ and the set $\{x_t \mid (\exists x_a, x_p)B(x_i, x_a, x_t, x_p)\}$ of titles given an ISBN number $x_i$, but we cannot obtain the set $\{\langle x_a, x_t\rangle \mid (\exists x_i, t_p)B(x_i, x_a, x_t, x_p)\}$ of authors and titles, given no input.

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

**Definition 3 (Executable)** $Q \in \text{CQ}^\neg$ 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 predicate. $Q \in \text{UCQ}^\neg$ with $Q := Q_1 \vee \ldots \vee Q_k$ is $\mathcal{P}$-*executable* if every $Q_i$ is $\mathcal{P}$-executable. We consider the empty rule $\bot$ to be executable and to return an empty result relation.

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

**Definition 4 (Orderable)** $Q \in \text{UCQ}^\neg$ with $Q := Q_1 \vee \ldots \vee Q_k$ is $\mathcal{P}$-*orderable* if for every $Q_i \in \text{CQ}^\neg$ 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.

**Example 4 (Orderable, Not Executable)** Given access patterns $\mathcal{P} := \{B^{\text{iooo}}, B^{\text{ooio}}, C^{\text{o}}, L^{\text{o}}\}$ the query

$$Q(x_i, x_a, x_t, x_p) \longleftarrow B(x_i, x_a, x_t, x_p), C(x_i), \neg L(x_t)$$

is not executable because $x_i$ appears first in an input slot. However

$$Q'(x_i, x_a, x_t, x_p) \longleftarrow C(x_i), B(x_i, x_a, x_t, x_p), \neg L(x_t)$$

is executable, so $Q$ is orderable.

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

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

**Example 5 (Feasible, Not Orderable)** Given access patterns $\mathcal{P} := \{B^{\text{iooo}}, B^{\text{ooio}}, C^{\text{o}}, L^{\text{o}}\}$ the query

$$Q(x_a) \longleftarrow B(x_i, x_a, x_t, x_p), L(x_t), B(y_i, x_a, y_t, y_p), C(y_i)$$

$$Q(x_a) \longleftarrow B(x_i, x_a, x_t, x_p), L(x_t), \neg B(y_i, x_a, y_t, y_p), C(y_i)$$

is not orderable since $y_t$ and $y_p$ cannot be bound, but is feasible because it is equivalent to

$$Q'(x_a) \longleftarrow L(x_t), B(x_i, x_a, x_t, x_p), C(y_i)$$

which is executable.

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

**Definition 7 (Answerable Part $A^Q$)** Given $Q \in \text{CQ}^\neg$, if $Q$ is unsatisfiable then $A^Q$ is $\bot$. If $Q$ is satisfiable, $A^Q$ is the query given by the $Q$-answerable literals in $Q$, in the order given by the algorithm ANSWERABLE (see 1). If $Q \in \text{UCQ}^\neg$ with $Q = Q_1 \vee \ldots Q_k$ then $A^Q = A^{Q_1} \vee \ldots \vee A^{Q_k}$.

Notice that the answerable part $A^Q$ of $Q$ is always executable, but may be empty.

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

**Proposition 2** *There is a quadratic-time algorithm for computing $A^Q$.*

The algorithm is given in Figure 1.

**Corollary 1** *There is a quadratic-time algorithm for checking whether $Q \in \text{UCQ}^\neg$ is orderable.*

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

**Proposition 3** *If $Q \in \text{UCQ}^\neg$, then $Q \sqsubseteq A^Q$.*

**Corollary 2** *If $Q \in \text{UCQ}^\neg$ and $A^Q \sqsubseteq Q$, then $Q$ is feasible.*

PROOF If $A^Q \sqsubseteq Q$ then $A^Q \equiv Q$ and therefore, since $A^Q$ is executable, $Q$ is feasible.

We will see in Section 5 (and in the appendix) that the converse holds as well; this is one of our main results. We need one more technical result for the algorithms and proofs presented in the next sections.

**Proposition 4** *$Q \in \text{CQ}^\neg$ is unsatisfiable iff there exists a predicate $R$ 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 $Q \in \text{CQ}^\neg$ is satisfiable in quadratic time: for every $R(\bar{x})$ in $Q^+$, look for $\neg R(\bar{x})$ in $Q^-$.

**Input**: – CQ$^\neg$ query $Q = L_1 \wedge \ldots \wedge L_k$ over relational schema $\mathbf{R}$ with access patterns $\mathcal{P}$
**Output**: – answerable part $A$ of $Q$ ($A^Q$)

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

**Fig. 1.** Algorithm ANSWERABLE(CQ$^\neg$)

## 4 Computing Plans and Answering Queries

Given a UCQ$^\neg$ query $Q = Q_1 \vee \cdots \vee Q_n$ over a relational schema $\mathbf{R}$ 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 ($\Pi_2^{\mathrm{P}}$-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 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 (quadratic in the size of the query) algorithm ANSWERABLE to compute $A^Q$, the answerable part of $Q$.

First we handle the special case that $Q$ is unsatisfiable. In this case we return "$\bot$", since $Q$ is equivalent to the empty query (the query returning no tuples). 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 $L_i$

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

procedure PLAN$^\star(Q)$
  for $i := 1$ to $n$ do
    $A_i := \text{ANSWERABLE}(Q_i, \mathcal{P})$
    $U_i := Q_i \setminus A_i$
    $Q_i^u := \begin{cases} A_i & \text{if } U_i = \emptyset \\ \bot & \text{otherwise} \end{cases}$
    $\bar{v} := \bar{x} \setminus \text{vars}(A_i)$
    $Q_i^o := A_i \wedge (\, \bar{v} = \overline{\texttt{null}}\,)$
  $Q^u := Q_1^u \vee \cdots \vee Q_n^u$
  $Q^o := Q_1^o \vee \cdots \vee Q_n^o$
  output $Q^u, Q^o$

**Fig. 2.** Algorithm PLAN$^\star$(UCQ$^\neg$)

that can be handled with the bindings $B$ we have so far (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 UCQ$^\neg$ query $Q$ (Figure 2). For each CQ$^\neg$ query $Q_i$ of $Q$, we compute its answerable part $A_i := A^{Q_i}$ and its unanswerable part $U_i$. As the underestimate of $Q_i^u$, we consider $A_i$ if $U_i$ is empty; else we dismiss $Q_i$ 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 PLAN$^\star$ is efficient, requiring at most quadratic time for any disjunct $Q_i$.

**Example 6 (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

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

procedure FEASIBLE($Q$)
  $(Q^u, Q^o) :=$ PLAN$^\star(Q)$
  if $Q^u = Q^o$ then return true
  else if $Q^o$ contains `null` then return false
  else return $Q^o \sqsubseteq Q$

**Fig. 3.** Algorithm FEASIBLE(UCQ$^\neg$)

the access pattern for $B$. Hence, we will end up with the following plans for $Q^u = Q_2^u$ and $Q^o = Q_1^o \vee Q_2^o$.

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

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

*Feasibility Test.* While PLAN$^\star$ is an efficient way to compute plans for a query $Q$, if it returns $Q^u \neq Q^o$ then we do not known whether $Q$ is feasible. One way, discussed below, is to not perform any static analysis in addition to PLAN$^\star$ 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 Consider Figure 3. First, FEASIBLE calls PLAN$^\star$ to compute the two plans $Q^u$ and $Q^o$. Ideally, $Q^u$ and $Q^o$ coincide, so feasibility is established. Similarly, if the overestimate contains some CQ$^\neg$ sub-query in which a `null` occurs, we know that $Q$ cannot be feasible. Otherwise, $Q$ may still be feasible, i.e., provided that $A^Q$ (= overestimate $Q^o$ if there are no `null`'s) is contained in $Q$. The complexity of FEASIBLE is dominated by the $\Pi_2^{\mathrm{P}}$-complete containment check $A^Q \sqsubseteq Q$.

### 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.

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

```
procedure ANSWER*(Q)
  (Q^u, Q^o) := PLAN*(Q)
  ans_u := ANSWER(Q^u, D)
  ans_o := ANSWER(Q^o, D)
  Δ := ans_o \ ans_u
  output ans_u
  if Δ = ∅ then output "answer is complete"
  else
      output "answer is not known to be complete"
      output "these tuples may be part of the answer:"
      output Δ
      if Δ has no null values then
        output "answer is at least" |ans_u|/|ans_o| "complete"
   /* optional: minimize Δ using dom on U_i */
```

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

Fortunately, the situation is not that bad after all. First, as indicated above, we may use the outcome of the efficient PLAN$^\star$ 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$^\star$ in Figure 4. We first let PLAN$^\star$ 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 $\Delta$ 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.

**Example 7 (Not Feasible, Runtime Complete)** Consider the plans created for the query in Example 6:

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

Given that $B^{ii}$ is the only access pattern for $B$, the query $Q_1$ in Example 6 is not feasible since we cannot create a binding for $B$'s $y$. However, for a given

database instance $D$,[6] 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.

Often it is not accidental that certain disjuncts evaluate to false, but rather it follows from some underlying semantic constraints, in which case the "cut-off" unanswerable residues do not compromise the completeness of the answer.

**Example 8 (Dependencies)** In our example above, 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 $Q_1^o(x, y)$ in Example 6 could have been 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.

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 $\Delta$ 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 $\Delta$ 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 `null` values in the overestimate though.

**Example 9 (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 the only thing we can definitely infer when we see $(\mathtt{a}, \mathtt{null})$ in the answer is 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

[6] Recall that behind $R$ and $S$ there may actually be web service operations. So what we call "the database" $D$ is really the union of the different parts of sources.

in this case we cannot give a numerical value for the completeness information in ANSWER$^\star$.

From Theorem 1 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$^\star$ 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$^\star$. However, in the case of infeasible queries, there are still additional improvements that can be made. Consider the algorithm PLAN$^\star$ 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 10 (Domain Enumeration)** For our running example from above, instead of discarding $Q_1^u$, 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: $\mathsf{dom}(x) \longleftarrow R(x, \_) \vee R(\_, x) \vee \ldots$

This domain enumeration approach has been used in various forms [DL97]. Note that in our setting of ANSWER$^\star$ we can create a very dynamic handling of answers: if ANSWER$^\star$ 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$^\star$ 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

Here we establish the complexity of determining whether a safe $Q \in$ UCQ$^\neg$ is feasible. We assume all queries we refer to are safe.

### 5.1 Query Containment

We will need to discuss the query containment problem for UCQ$^\neg$ queries. In general, query $P$ is said to be *contained* in query $Q$ (in symbols, $P \sqsubseteq Q$) if for every instance $D$, ANSWER$(P, D) \subseteq$ ANSWER$(Q, D)$.

**Definition 8 (Containment Problem)** 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 $P \sqsubseteq Q$.

**Definition 9 (Containment Mapping)** For $P, Q \in \text{CQ}$, a function $\sigma$: vars($Q$)$\rightarrow$ vars($P$) is a *containment mapping* if $P$ and $Q$ have the same free (distinguished) variables, $\sigma$ 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 5** *If $P, Q \in$ CQ then $P \sqsubseteq Q$ iff there is a containment mapping $\sigma$: vars($Q$) $\rightarrow$ vars($P$).*

**Proposition 6** *If $P, Q \in$ UCQ with $P = P_1 \vee \ldots \vee P_k$ and $Q = Q_1 \vee \ldots \vee Q_\ell$, then $P \sqsubseteq Q$ iff for every $i$ with $1 \leqslant i \leqslant k$ there is a $j$ with $1 \leqslant j \leqslant \ell$ so that $P_i \sqsubseteq Q_j$.*

**Proposition 7** *[CM77] Both* CONT(CQ) *and* CONT(UCQ) *are NP-complete.*

The problems CONT(CQ$^\neg$) and CONT(UCQ$^\neg$), which we will need, are harder.

**Proposition 8** *[SY80] [LS93] Both* CONT(CQ$^\neg$) *and* CONT(UCQ$^\neg$) *are $\Pi_2^{\text{P}}$-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 predicate 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 10 (Feasibility Problem)** We write FEASIBLE($\mathcal{L}, \mathcal{P}$) for the following decision problem: For a class of queries $\mathcal{L}$ and a set of access patterns $\mathcal{P}$, given $Q \in \mathcal{L}$ determine whether $Q$ is feasible.

The following theorem, from which our main results follow, is proved in the appendix.

**Theorem 1** *If $Q \in$ UCQ$^\neg$, $E$ is executable, and $Q \sqsubseteq E$, then $Q \sqsubseteq A^Q \sqsubseteq E$. That is, $A^Q$ is a minimal feasible query containing $Q$.*

**Corollary 3** *$Q$ is feasible iff $A^Q \sqsubseteq Q$.*

**Theorem 2**

$$\text{FEASIBLE}(\text{UCQ}^\neg, \mathcal{P}) \equiv_m^{\text{P}} \text{CONT}(\text{UCQ}^\neg)$$

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

Proof From Corollary 3 and Proposition 2 it follows that

$$\text{FEASIBLE}(\text{UCQ}^\neg, \mathcal{P}) \leqslant_m^{\text{P}} \text{CONT}(\text{UCQ}^\neg).$$

For the opposite direction, consider two queries $P, Q \in \text{UCQ}^\neg$ where $P := P_1 \vee \ldots \vee P_k$. We define the query

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

where $y$ is a variable not appearing in $P$ or $Q$ and $B$ is a predicate not appearing in $P$ or $Q$ with access pattern $B^{\text{i}}$. We give predicates $R$ appearing in $P$ or $Q$ output access patterns (i.e., $R^{\text{ooo}\cdots}$). 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, $A^{Q'} \equiv P \vee Q$. If $P \sqsubseteq Q$, then $A^{Q'} \equiv P \vee Q \equiv Q \sqsubseteq Q'$ so by Corollary 3, $Q'$ is feasible. If $P \not\sqsubseteq Q$, then since $P' \sqsubset P$ and $P' \not\sqsubseteq Q$ we have $A^{Q'} \equiv P \vee Q \not\sqsubseteq P' \vee Q \equiv Q'$ so again by Corollary 3, $Q'$ is not feasible. This shows

$$\text{CONT}(\text{UCQ}^\neg) \leqslant_m^{\text{P}} \text{FEASIBLE}(\text{UCQ}^\neg, \mathcal{P}).$$

Since $\text{CONT}(\text{UCQ}^\neg)$ is $\Pi_2^{\text{P}}$-complete, we have

**Corollary 4** $\text{FEASIBLE}(\text{UCQ}^\neg, \mathcal{P})$ *is* $\Pi_2^{\text{P}}$*-complete.*

The class of queries $\text{UCQ}^\neg$ includes the classes of queries CQ, UCQ, and $\text{CQ}^\neg$ (conjunctive queries, with union, and with negation, respectively). We have the following strict inclusions

$$\text{CQ} \subsetneq \text{UCQ}, \text{CQ}^\neg \subsetneq \text{UCQ}^\neg.$$

In the following subsections we show that the algorithm Feasible which essentially consists of the following two steps:

- compute $A^Q$
- test $A^Q \sqsubseteq Q$

provides optimal processing for all these subclasses of $\text{UCQ}^\neg$. Also, we compare algorithm Feasible to the algorithms given in [LC01].

### 5.3 Conjunctive Queries

Li and Chang [LC01] show that $\text{FEASIBLE}(\text{CQ}, \mathcal{P})$ 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 $A^M = M$ (they call this algorithm CQstable).
– Compute $A^Q$, then check that $A^Q \sqsubseteq Q$ (they call this algorithm CQstable*).

The advantage of the latter approach is that $A^Q$ may be equal to $Q$, eliminating the need for the equivalence check. For conjunctive queries, algorithm Feasible is exactly the same as CQstable*.

**Example 11 (CQ Processing)** Consider access patterns $F^{\text{o}}$ and $B^{\text{i}}$ and the CQ

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

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 $A := A^Q$

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

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, $\mathcal{P}$) is NP-complete and provide two algorithms for testing feasibility of $Q \in \text{UCQ}$ with $Q = Q_i \vee \ldots \vee Q_k$:

– Find a minimal (with respect to union) $M \in \text{UCQ}$ so $M \equiv Q$ with $M = M_i \vee \ldots \vee M_\ell$, 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 $Q_i$s, 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 $A^Q$ may be equal to $Q$, eliminating the need for the equivalence check.

**Example 12 (CQU Processing)** Consider access patterns $F^{\text{o}}$, $G^{\text{o}}$, $H^{\text{o}}$, and $B^{\text{i}}$ and the query

$$Q(x) \longleftarrow F(x), G(x)$$
$$Q(x) \longleftarrow F(x), H(x), B(y)$$
$$Q(x) \longleftarrow F(x)$$

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 $Q$

$$P(x) \longleftarrow F(x), G(x)$$

$$P(x) \longleftarrow F(x)$$

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

$$A(x) \longleftarrow F(x), G(x)$$

$$A(x) \longleftarrow F(x), H(x)$$

$$A(x) \longleftarrow F(x)$$

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

### 5.5 Conjunctive Queries with Negation

**Proposition 9** $\text{CONT}(\text{CQ}^\neg) \leqslant_m^{\text{P}} \text{FEASIBLE}(\text{CQ}^\neg)$

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

$$P(\bar{x}) :\equiv (\exists \bar{x}_0)(\hat{R}_1(\bar{x}_1) \wedge \ldots \wedge \hat{R}_k(\bar{x}_k))$$

and

$$Q(\bar{x}) :\equiv (\exists \bar{y}_0)(\hat{S}_1(\bar{y}_1) \wedge \ldots \wedge \hat{S}_\ell(\bar{y}_\ell))$$

where the $R_i$s and $S_i$s are not necessarily distinct and the $x_i$s and $y_i$s are also not necessarily distinct. Then define

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

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

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

and therefore

$$P \sqsubseteq Q \text{ iff } P \sqsubseteq P \wedge Q \text{ iff } A^L \sqsubseteq L \text{ iff } L \text{ is feasible.}$$

The second iff follows from the fact that every containment mapping $\eta$: $P \wedge Q \longrightarrow P$ corresponds to a unique containment mapping $\eta'$: $L \longrightarrow A^L$ and vice versa.

Since $\text{CONT}(\text{CQ}^\neg)$ is $\Pi_2^{\text{P}}$-complete, we have

**Corollary 5** $\text{FEASIBLE}(\text{CQ}^\neg, \mathcal{P})$ *is* $\Pi_2^{\text{P}}$*-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 ($UCQ^\neg$). From a theoretical point of view, our main theorem (Theorem 2) shows that checking feasibility for $CQ^\neg$ and $UCQ^\neg$ is equivalent to checking containment for $CQ^\neg$ and $UCQ^\neg$ (respectively) and thus $\Pi_2^P$-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 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$^\star$), and defer the containment check in the algorithm FEASIBLE if possible, and (ii) use a novel runtime algorithm ANSWER$^\star$, 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 "sandwiching" the result from above and below is not applicable (essentially, the underestimate is always empty when not considering union).

While some of our work may seem rather technical in nature, it is in fact motivated and driven by a number of very practical engineering problems. As part of the Bioinformatics Research Network project [BIR], 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 an $UCQ^\neg$ plan. We have used ANSWERABLE and a simplified version (without the containment check) of PLAN$^\star$ and ANSWER$^\star$ in the system. Our theoretical investigations have solved several of the pending algorithmic issues and we can now proceed with the extension of the mediator planner. In the BIRN project, as in many other projects pertaining to scientific data integration, sources with limited query capabilities are ubiquitous: in addition to relational and XML databases, many data and computational resources are accessible through web services which – as we have shown – can be conceived of as very limited relational sources with access patterns.

For example, in the SEEK and SciDAC projects [SEE,SDM02] we are building distributed scientific workflow systems which can be seen as procedural variants of the declarative query plans which a mediator is processing.

Consider, e.g., the problem of a molecular biologist who is interested in finding co-regulated genes, based on sequence similarity and cluster analysis using various online databases and computational tools. Increasingly these tools and databases become accessible as web services and thus within the realm of query

planning with limited access patterns. The query rewriting techniques that we are employing for these kinds of scientific data workflows [LAG03] contain the presented problem of computing executable plans and thus can benefit from our results.

In this paper, we have not dealt with the optimization problem of how to select among the several possible executable plans. For example, how should we "bundle" together sub-plans coming from the same source? Clearly, these are important questions and extensions of the current work and we are planning to address these issues in the future. Similarly, 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, NSF-ITR 0225676, NSF-ITR 0225673, and DOE/SciDAC DE-FC02-01ER25486.

## References

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

[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.

[LAG03] B. Ludäscher, I. Altintas, and A. Gupta. Compiling Abstract Scientific Workflows into Web Service Workflows. In *15th Intl. Conference on Scientific and Statistical Database Management (SSDBM)*, Boston, Massachussets, 2003.

[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*, chapter 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*, 2003. conditional acceptance.

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

[NL03] A. Nash and B. Ludäscher. From Feasible Queries to Executable Plans – Complexity and Algorithms. Technical Report BIRN-TR-07-03, Bioinformatics Research Network, U.C. San Diego, 2003.

[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.

[SDM02] Scientific Data Management Center (SDM). `http://sdm.lbl.gov/sdmcenter/` and `http://www.er.doe.gov/scidac/`, 2002.

[SEE] Science Environment for Ecological Knowledge (SEEK). `http://seek.ecoinformatics.org/`.

[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.

[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.

# A Appendix: Auxiliary Material and Proofs

We assume all queries we refer to are safe. In particular, Theorems 3 and 4 below hold only for safe queries. We will need the following results.

**Proposition 10** *If $\hat{R}(\bar{x})$ is $Q$-answerable, then it is $Q^+$ answerable.*

**Proposition 11** *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 10. 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 $P^R$s is executable and consists of $\hat{S}(\bar{x})$ and literals from $P$. That is, $\hat{S}(\bar{x})$ is $P$-answerable.

vars($Q$) is the set of variables that appear in $Q$.

**Proposition 12** *If $P, Q \in \mathrm{CQ}$, $\sigma$: vars($Q$)$\rightarrow$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.

Here we give the proof of theorem 1. We will need the following results.

Given $P, R \in \mathrm{CQ}^\neg$ where $P :\equiv (\exists\bar{x})P'$ and $Q :\equiv (\exists\bar{y})Q'$ with $P', Q'$ quantifier free (i.e., consisting 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 (2 and 5).

**Theorem 3** *[WL03] If $P, Q \in \mathrm{CQ}^\neg$ then $P \sqsubseteq Q$ iff $P$ is unsatisfiable or there is a containment mapping $\sigma$: $\mathrm{vars}(Q) \to \mathrm{vars}(P)$ 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 $P, R(\sigma\bar{y}) \sqsubseteq Q$.*

Proof (Proposition 3) For $Q \in \mathrm{CQ}$ this is clear since $A^Q$ contains only literals from $Q$ and therefore the identity map is a containment mapping from $A^Q$ to $Q$. If $Q \in \mathrm{CQ}^\neg$ and $Q$ is unsatisfiable, the result is obvious. Otherwise the identity is a containment mapping from $(A^Q)^+$ to $Q^+$. If a negative literal $\neg R(\bar{y})$ appears in $(A^Q)^+$, then since $\neg R(\bar{y})$ also appears in $Q$, $Q, R(\bar{y})$ is unsatisfiable, and therefore $Q \sqsubseteq A^Q$.

**Theorem 4** *[WL03] 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 $\sigma$: $\mathrm{vars}(Q_i) \to \mathrm{vars}(P)$ 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 1.

**Lemma 1** *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 $\sigma$ 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 10, $\neg N_{m+1}(\bar{y})$ is $E_s^+$-answerable. Therefore by Proposition 12, $\neg N_{m+1}(\bar{x})$ is $Q^+, N_1(\bar{x}_1), \ldots, N_m(\bar{x}_m)$-answerable and by the induction hypothesis, $Q^+$-answerable. Therefore, by Proposition 11 and the induction hypothesis, $\hat{R}(\bar{x})$ is $Q^+$-answerable.

**Lemma 2** *If $Q, N_1(\bar{x}_1), \ldots, N_m(\bar{x}_m)$ is unsatisfiable, then $A^Q, N_1(\bar{x}_1), \ldots, N_m(\bar{x}_m)$ 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 4 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 10, $\neg N_i(\bar{y})$ is $E_s^+$-answerable. Therefore by Proposition 12, $\neg N_i(\bar{x}_i)$ is $Q^+, N_1(\bar{x}_1), \ldots, N_m(\bar{x}_m)$-answerable and by Lemma 1, $Q^+$-answerable. Therefore, we must have $\neg N_i(\bar{x}_i)$ in $A^Q$, so $A^Q, N_1(\bar{x}_1), \ldots, N_m(\bar{x}_m)$ is also unsatisfiable.

We are now ready to prove one of our main theorems.

**Theorem 1** *For $Q \in \mathrm{UCQ}^\neg$ with $Q = Q_1 \vee \ldots \vee Q_k$ then if $E$ is executable and $Q \sqsubseteq E$ then $Q \sqsubseteq A^Q \sqsubseteq E$. That is, $A_Q$ is a minimal feasible query containing $Q$.*

PROOF We have $Q \sqsubseteq A^Q$ from Proposition 3. Set $A_i = A^{Q_i}$. 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 $A^Q \sqsubseteq E$.

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 theorem 4.3 in [WL03] 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 2 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

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

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 10, since $E_s$ is executable, $R(\bar{y})$ is $E_s^+$-answerable. By Proposition 12, $R(\bar{x})$ is $Q_i^+, N_1(\bar{x}_1), \ldots, N_m(\bar{x}_m)$-answerable and so, by Lemma 1, $Q_i^+$-answerable. Therefore, $R(\bar{x})$ is in $A_i^+$, which is a contradiction.