Rewriting Queries Using Views with Access Patterns Under Integrity Constraints*

Alin Deutsch[1], Bertram Ludäscher[2], and Alan Nash[3]

University of California, San Diego
[1]deutsch@cs.ucsd.edu, [2]ludaesch@sdsc.edu, [3]anash@math.ucsd.edu

Abstract. We study the problem of rewriting queries using views in the presence of access patterns, integrity constraints, disjunction, and negation. We provide asymptotically optimal algorithms for finding minimal containing and maximal contained rewritings and for deciding whether an exact rewriting exists. We show that rewriting queries using views in this case reduces (a) to rewriting queries with access patterns and constraints without views and also (b) to rewriting queries using views under constraints without access patterns. We show how to solve (a) directly and how to reduce (b) to rewriting queries under constraints only (semantic optimization). These reductions provide two separate routes to a unified solution for all three problems, based on an extension of the relational chase theory to queries and constraints with disjunction and negation. We also handle equality and arithmetic comparisons.

1 Introduction

We study the problem of rewriting a query Q in terms of a given set of views $\mathcal{V}$ with limited access patterns $\mathcal{P}$, under a set Σ of integrity constraints. More precisely, we are interested in determining whether there exists a query plan Q', expressed in terms of the views $\mathcal{V}$ only, that is executable (i.e., observes $\mathcal{P}$) and equivalent to Q for all databases satisfying Σ. We say that Q is *feasible* if such Q' exists. For infeasible Q we seek the minimal containing and maximal contained executable queries, which provide the "best possible" executable query plans for approximating the answer to Q from above and below. Our results unify and extend a number of previous results in data integration (see related work). In particular, they apply to queries, views, and constraints over unions of conjunctive queries with negation ($\text{UCQ}^\neg$), equality and arithmetic comparisons.

The following example shows the common case of a query that has no equivalent executable rewriting (i.e., is not feasible) in the absence of constraints, but that can yield such a rewriting when constraints are given.

Example 1. Consider the following set of relations with access patterns: conference $C^{\text{io}}(a,t)$, journal $J^{\text{io}}(a,t)$, magazine $M^{\text{oo}}(a,t)$, PC-magazine $P^{\text{ioo}}(a,t,p)$, the set of listed publishers $L^{\text{i}}(p)$, repository $R^{\text{oo}}(a,t)$, ACM anthology $A^{\text{iii}}(a,t,o)$, and

* Supported by NSF/CAREER 0347968, NSF/ITR 0225673 (GEON), NSF/ITR 0225674 (SEEK), DOE SciDAC DE-FC02-01ER25486 (SDM), NIH/NCRR 1R24 RR019701-01 Biomedical Informatics Research Network Coordinating Center (BIRN-CC).

DBLP conference article $D^{\text{ooo}}(a, t, c)$. The relation symbols are annotated with access patterns, indicating which arguments must be given as inputs (marked 'i') and which ones can be retrieved as outputs (marked 'o') when accessing the relation. For example $C^{\text{io}}(a, t)$ means that an author a has to be given as input before one can retrieve the titles t of a's conference publications from $C(a, t)$.

Let Q be the query which asks for pairs of authors and titles of conference publications, journal publications, and magazines which are not PC-magazines:

$$Q(a,t) \leftarrow \underline{C(a,t)} \tag{1}$$
$$Q(a,t) \leftarrow \underline{J(a,t)} \tag{2}$$
$$Q(a,t) \leftarrow M(a,t), \underline{\neg P(a,t,p)}, \underline{L(p)} \tag{3}$$

Q cannot be executed since no underlined literal is answerable: e.g., the access patterns require a to be bound before invoking $C(a, t)$ but no such binding is available. Worse yet, Q is not even feasible, i.e., there is no executable query Q' equivalent to Q. However, if the following set Σ of integrity constraints is given, an executable Q' can be found that is equivalent under Σ:

$$\forall a \forall t\ C(a,t) \rightarrow \exists c\ D(a,t,c) \tag{4}$$
$$\forall a \forall t\ J(a,t) \rightarrow \exists p\ R(a,t) \wedge \neg P(a,t,p) \wedge L(p) \vee\ \exists o \exists c\ A(a,t,o), D(a,t,c) \tag{5}$$
$$\forall a \forall t\ M(a,t) \rightarrow \exists p\ \neg P(a,t,p), L(p) \tag{6}$$

Constraint (4) states that every conference publication is a DBLP conference publication; (5) states that every journal publication is available from a repository, comes from a listed publisher and is not a PC magazine, or is available from the ACM anthology and from DBLP; and (6) states that magazine articles are not PC-magazine articles. We are only interested in databases which satisfy these constraints Σ. On those databases, Q is equivalent to Q^{Σ}, obtained by "chasing" Q with Σ:

$$Q^{\Sigma}(a,t) \leftarrow C(a,t), D(a,t,c)$$
$$Q^{\Sigma}(a,t) \leftarrow J(a,t), R(a,t), \underline{\neg P(a,t,p)}, \underline{L(p)}$$
$$Q^{\Sigma}(a,t) \leftarrow J(a,t), \underline{A(a,t,o)}, D(a,t,c)$$
$$Q^{\Sigma}(a,t) \leftarrow M(a,t), \underline{\neg P(a,t,p)}, \underline{L(p)}$$

Again, unanswerable literals are underlined. The *answerable part* $\text{ans}(Q^{\Sigma})$ is obtained (roughly) by removing unanswerable parts:

$$\text{ans}(Q^{\Sigma})(a,t) \leftarrow C(a,t), D(a,t,c) \tag{7}$$
$$\text{ans}(Q^{\Sigma})(a,t) \leftarrow J(a,t), R(a,t) \tag{8}$$
$$\text{ans}(Q^{\Sigma})(a,t) \leftarrow J(a,t), D(a,t,c) \tag{9}$$
$$\text{ans}(Q^{\Sigma})(a,t) \leftarrow M(a,t) \tag{10}$$

In general, the answerable part is not equivalent to Q: e.g., the subquery (10) is not contained in (3) and thus $\text{ans}(Q^{\Sigma})$ might produce more answers than Q. However

the equivalence may still hold under Σ, i.e., for all databases satisfying Σ. This can be checked (cf. Corollary 2) and is indeed the case here. Then $\text{ans}(Q^{\Sigma})$ is the desired executable plan, equivalent to Q for all databases satisfying the constraints Σ.

As we will show, if there is an equivalent query Q' under Σ, our algorithm will find it, and if no such Q' exists, we can find the minimal containing and the maximal contained plans, providing least overestimate and greatest underestimate queries for Q under Σ, respectively.

Example 2. This example illustrates that our techniques can also rewrite queries in terms of views with access patterns. For example, the rules

$$V_1^{\text{oo}}(a,t) :- C(a,t), R(a,t)$$
$$V_2^{\text{io}}(a,t) :- C(a,t), \neg R(a,t)$$

state that the view V_1 has conference articles that are also in the repository R, while V_2 has those that are not in R. The access patterns indicate that at least a must be given when accessing $V_2(a,t)$, while no inputs are required for accessing V_1. We will show that if we want to rewrite a query in terms of the views only, this can be achieved by considering constraints and access patterns only. To this end, we model views as constraints and also include "negation constraints" of the form $\forall a \forall t\,(\texttt{true} \rightarrow (R(a,t) \vee \neg R(a,t)))$. Chasing the query $Q(a,t) \leftarrow C(a,t)$ with the latter yields

$$Q'(a,t) \leftarrow C(a,t), R(a,t)$$
$$Q'(a,t) \leftarrow C(a,t), \neg R(a,t)$$

which then rewrites in terms of V_1 and V_2 to

$$Q''(a,t) \leftarrow V_1(a,t)$$
$$Q''(a,t) \leftarrow V_2(a,t).$$

Here, Q'' is not executable (the access pattern for V_2 requires a to be bound). Under the constraint $\forall a \forall t\,(C(a,t) \rightarrow R(a,t))$, our algorithm can discard the unanswerable second rule, resulting in the executable rewriting $Q'''(a,t) \leftarrow V_1(a,t)$.

Contributions. We solve the problem of rewriting queries using views with limited access patterns under integrity constraints (denoted $\{Q, \mathcal{V}, \mathcal{P}, \Sigma\}$) and prove that feasibility is **NP**-complete for queries, views, and constraints over[1] UCQ and $\mathbf{\Pi_2^P}$-complete for UCQ$^{\neg}$. These results hold in those cases when the chase terminates and its result is not too large (Theorem 10). A fairly general sufficient condition for this (undecidable) behavior of the chase is given by Theorem 8. We present an algorithm, ViewRewrite, which is guaranteed to find an exact plan (if one exists) or at least the minimal containing plan (unique if it exists) (Theorem 9). We also give an algorithm for finding the maximal contained executable plan (Theorem 11).

[1] A constraint *over* $\mathcal{L}$ is an implication $\forall \bar{x}(U \rightarrow V)$ with $U, V \in \mathcal{L}$ (cf. Section 2).

One side effect of our results is a unified treatment for three flavors of rewriting problems which have been introduced and solved separately. We show that $\{Q, \mathcal{V}, \mathcal{P}, \Sigma\}$ reduces to $\{Q, \mathcal{P}, \Sigma\}$, i.e., rewriting queries with access patterns and constraints without views (Theorem 9) and also to $\{Q, \mathcal{V}, \Sigma\}$, i.e. rewriting queries under constraints using views without access patterns (Theorem 15).

We show how to solve $\{Q, \mathcal{P}, \Sigma\}$ and $\{Q, \mathcal{V}, \Sigma\}$ by reduction to rewriting queries under constraints only (semantic optimization, denoted $\{Q, \Sigma\}$). These reductions provide two separate routes $\{Q, \mathcal{V}, \mathcal{P}, \Sigma\} \rightsquigarrow \{Q, \mathcal{P}, \Sigma\} \rightsquigarrow \{Q, \Sigma\}$ and $\{Q, \mathcal{V}, \mathcal{P}, \Sigma\} \rightsquigarrow \{Q, \mathcal{V}, \Sigma\} \rightsquigarrow \{Q, \Sigma\}$ to a unified solution for all three problems, based on our extension of the relational chase theory to queries and constraints with disjunction and negation. Specifically we show that a minimal containing query in the $\{Q, \mathcal{P}, \Sigma\}$ case can be obtained by chasing Q with Σ and computing the answerable part. Similarly, in the presence of views, we can compute the minimal containing query by chasing with Σ and the constraints corresponding to $\mathcal{V}$ and again computing the answerable part.

We also extend the above results to handle equality and arithmetic comparisons by modeling them with constraints (Section 7).

Related Work. There is a large body of related work that deals with one or more of the following three aspects: (i) query rewriting under *limited access patterns*, see [22, 20, 14, 9, 23, 16, 15, 19, 18] and references within; (ii) query rewriting under *integrity constraints* (a.k.a. semantic query optimization), see for instance [12, 5] and references within; and (iii) query rewriting and answering *using views* [6, 7, 11]. These all have important applications in data integration and query optimization [13, 17, 10]. All of the above mentioned work on rewriting has focused on either of two flavors: *maximal contained* or *exact* rewritings.

In this paper, we introduce algorithms which deal uniformly with all three aspects of rewriting and find exact, maximal contained and *minimal containing* rewritings.

In the category of maximal contained rewritings, the closest related results are those of [7], which considers the most expressive queries and views, and of [12], which handles the most expressive constraints. [7] shows how to obtain a maximal contained rewriting for recursive Datalog queries using conjunctive query views. [7] also considers access patterns on the views as well as very restricted constraints (which can express the standard key but not all foreign key constraints) and it shows how to construct a recursive plan which is guaranteed to be maximal contained. As opposed to [7], we do not consider recursive queries but we allow negation and disjunction in queries, views and constraints (our constraints express key, foreign key, join, multi-valued, and embedded dependencies and beyond). Moreover, we provide decision procedures for the existence of an *exact* plan and, in its absence, we show how to obtain not only the best *contained* but also the best *containing* approximations. [12] finds the maximal contained rewriting of CQ queries under more expressive constraints than [7] (embedded dependencies), provided the predicate dependency graph is acyclic. However, views, access patterns and negation (in either query or constraints) are not handled.

With respect to finding exact rewritings, [5] shows how to treat views and integrity constraints uniformly for UCQ queries. The present paper extends these results to $\text{UCQ}^\neg$ queries, constraints, views with limited access patterns, and maximal contained and minimal containing rewritings. [16, 15] shows **NP**-completeness for deciding feasibility of UCQ queries over relations with limited access patterns (i.e. no negation, no views and no constraints are considered). Still in the absence of views and constraints, [19] shows that if negation is added then deciding feasibility becomes $\mathbf{\Pi_2^P}$-complete; [18] further extends the notion of feasibility to all first-order queries and characterizes the complexity of many first-order query classes.

Outline. The preliminaries in Section 2 include earlier results on containment and feasibility under access patterns. Section 3 presents our results on feasibility and rewriting with access patterns under constraints. In Section 4 we generalize these results to include views. In Section 5 we establish our results on maximal contained executable queries. Section 6 provides an alternative method for deciding feasibility: Instead of handling access patterns via the answerable part, we show that they too can be reduced to constraints and the chase. Section 7 shows how other extensions such as equality and arithmetic comparisons can all be treated uniformly via constraints.

2 Preliminaries

Queries. A *term* is a variable or constant. By $\bar{x}$ we denote a finite sequence of terms $x_1, \ldots, x_k$. We use lowercase letters $x, y, z, \ldots$ for terms and uppercase letters $P, Q, R, \ldots$ for relation symbols and queries. A *datalog rule* is an expression of the form $P(\bar{z}) \leftarrow \ell_1(\bar{x}_1), \ldots, \ell_n(\bar{x}_n)$ where each $\ell_i(\bar{x}_i)$ in the rule *body* is a *literal*, i.e., a *positive* atom $R(\bar{x})$ or a *negative* literal $\neg R(\bar{x})$. Given a rule Q, we define $\text{head}(Q)$ and $\text{body}(Q)$ to be the parts to the left and to the right of the arrow, respectively. A datalog *program* is a finite set of datalog rules. We only consider nonrecursive programs and we further require that all rules have the same head. Therefore, $\text{head}(P)$ is well-defined for the programs P we consider.

We represent queries by programs unless otherwise specified. If a query Q is given by multiple rules $Q_1, \ldots, Q_n$, we denote this by $Q = \bigvee_i Q_i$ and we have $Q(\mathcal{D}) = \bigcup_i Q_i(\mathcal{D})$, where $Q(\mathcal{D})$ denotes the result of query Q on database $\mathcal{D}$.

Queries given by a single rule are *conjunctive queries* (CQ) if all literals are positive and *conjunctive queries with negation* ($\text{CQ}^\neg$) otherwise. Queries given by multiple rules are *unions of conjunctive queries* (UCQ) if all literals are positive and *conjunctive queries with negation* ($\text{UCQ}^\neg$) otherwise.

A query $Q \in \text{CQ}^\neg$ is *safe* if every variable which appears in the rule (whether in the head or in the body) appears positively in its body. A query $Q = \bigvee_i Q_i$ with $Q_1, \ldots, Q_n \in \text{CQ}^\neg$ is *safe* if every Q_i is safe. In the definition of $\text{ans}(Q)$ below, we will need to consider two special kinds of queries. A query $Q \in \text{CQ}^\neg$ given by $\text{head}(Q) \leftarrow \bot$ is unsatisfiable and is always safe (this is an extension of the definition above). A query $Q \in \text{CQ}^\neg$ given by a rule with an empty body is safe if there are no variables in the head (i.e., if the query is boolean).

Unless otherwise specified, all queries are $\text{UCQ}^\neg$ and safe. Furthermore, E, P, and Q always denote queries.

Containment. P is *contained* in Q ($P \sqsubseteq Q$) if, for all databases $\mathcal{D}$, $P(\mathcal{D}) \subseteq Q(\mathcal{D})$. P is *equivalent* to Q ($P \equiv Q$) if $P \sqsubseteq Q$ and $Q \sqsubseteq P$. Given a set of constraints Σ, P is *Σ-contained* in Q ($P \sqsubseteq_\Sigma Q$) if, for all $\mathcal{D}$ which satisfy Σ, $P(\mathcal{D}) \subseteq Q(\mathcal{D})$. P is *Σ-equivalent* to Q ($P \equiv_\Sigma Q$) if $P \sqsubseteq_\Sigma Q$ and $Q \sqsubseteq_\Sigma P$.

$\text{CONT}(\mathcal{L})$ is the decision problem: for $P, Q \in \mathcal{L}$ determine whether $P \sqsubseteq Q$ ($\mathcal{L}$ is a class of queries). $\text{CONT}_\Sigma(\mathcal{L})$ is the problem: for $P, Q \in \mathcal{L}$ decide whether $P \sqsubseteq_\Sigma Q$.

Theorem 1. *a)* $\text{CONT}(\text{CQ})$ *and* $\text{CONT}(\text{UCQ})$ *are* **NP***-complete [2].*
b) $\text{CONT}(\text{CQ}^\neg)$ *and* $\text{CONT}(\text{UCQ}^\neg)$ *are* $\mathbf{\Pi_2^P}$*-complete [21].*

Access Patterns. An *access pattern* for a k-ary relation R is an expression R^α where α is word of length k over the alphabet $\{\text{i}, \text{o}\}$. 'i' denotes a required *input slot* and 'o' denotes an *output slot* (no value required). Given access patterns $\mathcal{P}$, an *annotation* of Q assigns to each occurrence of a relation symbol a pattern from $\mathcal{P}$.

Definition 1 (Executable). Q is *executable* if it can be annotated so that every variable of a rule appears first positively in an output slot in the body of that rule.

Definition 2 (Feasible). Q is *feasible* if it is equivalent to an executable query Q'. $\text{FEASIBLE}(\mathcal{L})$ is the decision problem: for $Q \in \mathcal{L}$, determine whether Q is feasible.

For $Q \in \text{CQ}^\neg$, we say that a literal $\ell(\bar{x})$ (not necessarily in Q) is *Q-answerable* if there is an executable $Q' \in \text{CQ}^\neg$ which is a conjunction of $\ell(\bar{x})$ and literals in Q. The *answerable part* of a query Q is another query $\text{ans}(Q)$ defined below. $\text{ans}(Q)$ may be undefined for some queries Q, but when defined it is executable.

Definition 3 (Answerable Part). If $Q \in \text{CQ}^\neg$ is unsatisfiable we set the body of $\text{ans}(Q)$ to $\bot$; otherwise we set the body of $\text{ans}(Q)$ to the conjunction of the Q-answerable literals in Q in the order specified by the algorithm in the proof of Lemma 1. We set $\text{head}(\text{ans}(Q)) := \text{head}(Q)$. However, if the resulting query $\text{ans}(Q)$ is unsafe, we say that $\text{ans}(Q)$ is undefined. If $Q = \bigvee_i Q_i$ with $Q_1, \ldots, Q_n \in \text{CQ}^\neg$, we set $\text{ans}(Q) := \bigvee_i \text{ans}(Q_i)$. In this case $\text{ans}(Q)$ is defined iff every $\text{ans}(Q_i)$ is defined.

Lemma 1. $\text{ans}(Q)$ *can be computed in quadratic time.*

Proof. We consider the case when $Q \in \text{CQ}^\neg$; the case $Q \in \text{UCQ}^\neg$ is handled the same way, one rule at a time. Give $\text{ans}(Q)$ the same head as Q and build its body one literal at a time as follows. Start with $\mathcal{B}$, the set of bound variables, empty. Find the first literal $\ell(\bar{x})$ in Q not yet added to $\text{ans}(Q)$ such that,

- $\ell(\bar{x})$ is positive and there is some access pattern for it in $\mathcal{P}$ such that all variables in $\bar{x}$ which appear in input slots in $\ell(\bar{x})$ are in $\mathcal{B}$, or

- $\ell(\bar{x})$ is negative and its variables are in $\mathcal{B}$.

If there is no such literal, stop. Otherwise, add $\ell(\bar{x})$ to $\text{ans}(Q)$, set $\mathcal{B} := \mathcal{B} \cup \{\bar{x}\}$, and repeat. Clearly, this algorithm adds to the body of $\text{ans}(Q)$ all the Q-answerable literals in Q and no others.

The main results on testing feasibility for $\text{UCQ}^\neg$ queries are [19]: if defined, $\text{ans}(Q)$ is the minimal executable query containing Q (Theorem 2); checking feasibility of $\text{UCQ}^\neg$ queries can be reduced to checking $\text{UCQ}^\neg$ query containment (Corollary 1), and is in fact as hard as checking query containment of $\text{UCQ}^\neg$ queries (Theorem 3b). Checking feasibility of UCQ queries is **NP**-complete (Theorem 3a) [20].

Theorem 2. *If $Q \sqsubseteq E$ and E is executable then $\text{ans}(Q)$ is defined and $Q \sqsubseteq \text{ans}(Q) \sqsubseteq E$.*

Corollary 1. *Q is feasible iff $\text{ans}(Q)$ is defined and $\text{ans}(Q) \sqsubseteq Q$.*

Theorem 3. *a)* FEASIBLE(UCQ) *is* **NP***-complete.*
b) FEASIBLE($\text{UCQ}^\neg$) *is* $\mathbf{\Pi_2^P}$*-complete.*

Chase. We consider constraints of the form $\text{IC}(\mathcal{L}) := \{\, \forall \bar{x}\, (U \rightarrow V) \mid U, V \in \mathcal{L}\}$ where $\bar{x}$ is the set of free variables in both U and V. Such constraints express the containment of U in V and are known as *embedded dependencies* when $\mathcal{L} = \text{CQ}$.

Unless otherwise specified, we assume all constraints are subsets of $\text{IC}(\text{UCQ}^\neg)$. Furthermore Σ always denotes a set of constraints.

Given a set of constraints $\Sigma \subseteq \text{IC}(\text{UCQ})$, there is a well known procedure for extending a query $Q \in \text{UCQ}$ to another query Q' by an iterative procedure known as the *chase* which depends on the order O of the constraints. That is, we set $Q' := \text{chase}(Q, \Sigma, O) \in \text{UCQ}$. In [3] we extend this procedure to $Q \in \text{UCQ}^\neg$ and $\Sigma \subseteq \text{IC}(\text{UCQ}^\neg)$.

The chase does not always terminate (even in the case with no negation) and its syntactic form depends on the order O. However if the chase terminates for any two orders O_1 and O_2, then $\text{chase}(Q, \Sigma, O_1) \equiv \text{chase}(Q, \Sigma, O_2)$.

Definition 4 (Negation Constraints). $\Sigma^\tau_\neg \subseteq \text{IC}(\text{UCQ}^\neg)$ is the smallest set of constraints which contains, for each k, each k-ary relation R in the schema τ and some k-tuple $\bar{x}$ of variables, the constraint $\forall \bar{x}\, (\texttt{true} \rightarrow (\, R(\bar{x}) \vee \neg R(\bar{x})\,))$.

We allow unsafe sentences as constraints (we need them for $\Sigma^\tau_\neg$); however if Q is safe, then $\text{chase}(Q, \Sigma, O)$ is also safe, even when Σ includes unsafe sentences.

Definition 5 (Chase Result Q^Σ). $Q^\Sigma := \text{chase}(Q, \Sigma \cup \Sigma^\tau_\neg, O)$ for some order on which the chase terminates (if there is such order).

Notice that Q^Σ is defined only up to equivalence. The following two results extend previous results which do not handle negation.

Theorem 4 (Chase Completeness). *If Q^Σ is defined, then $Q^\Sigma \sqsubseteq P$* iff *$Q \sqsubseteq_\Sigma P$.*

Theorem 5. *If $\Sigma \subseteq \text{IC}(\text{UCQ}^\neg)$ and there is a polynomial p such that for all Q, $Q^\Sigma (= \bigvee_i Q'_i) \in \text{UCQ}^\neg$ and all i: $|Q'_i| \leqslant p(|Q|)$, then $\text{CONT}_\Sigma(\text{UCQ}^\neg)$ is* $\mathbf{\Pi_2^P}$*-complete.*

We write $Q^{\Sigma,\Sigma'}$ for $(Q^\Sigma)^{\Sigma'}$ which in general is not equivalent to $Q^{\Sigma \cup \Sigma'}$.

3 Integrity Constraints

We consider $\{Q, \mathcal{P}, \Sigma\}$, i.e., the problem of answering a query Q in the presence of access patterns $\mathcal{P}$ and integrity constraints Σ.

Definition 6 (Σ-Feasible). Q is *Σ-feasible* if it is Σ-equivalent to an executable query Q'. FEASIBLE$_\Sigma(\mathcal{L})$ is the decision problem: for $Q \in \mathcal{L}$, decide whether Q is Σ-feasible.

The main results in this section are that, if defined, ans(Q^Σ) is the minimal executable query Σ-containing Q (Theorem 6), that checking Σ-feasibility of UCQ$^\neg$ queries can be reduced to checking containment of UCQ$^\neg$ queries (Corollary 2), and that in those cases where Q^Σ is well-defined (i.e., the chase terminates) and not too large its complexity is the same as that of checking containment of UCQ$^\neg$ queries (Theorem 7b). Corresponding results hold for CQ, CQ$^\neg$, and UCQ (Theorem 7a). We outline the algorithms REWRITE and FEASIBLE which use the following functions:

- ans(Q), which given a query Q, produces the query ans(Q). A quadratic time algorithm for this function is outlined in the proof of Lemma 1.
- chase(Q, Σ, O), which given a query Q, a set of constraints Σ, and an order on the constraints O, produces the query chase(Q, Σ, O). An algorithm for this function is outlined in [3]. No guarantees are given for the running time or space of chase(Q, Σ, O); in fact, it may not even terminate.
- contained(P, Q), which given queries P and Q, returns true if $P \sqsubseteq Q$, false otherwise (its complexity is given in Theorem 1).

Note that algorithm REWRITE(Q, Σ) may return `undefined` or may not terminate; similarly, FEASIBLE(Q, Σ) may not terminate. Theorem 6 and Corollary 2 below show that algorithms REWRITE and FEASIBLE are correct and complete, in those cases in which the chase terminates regardless of the order O.

function REWRITE(Q, Σ)
(1) Compute $\Sigma' := \Sigma \cup \Sigma^\tau_\neg$ and pick some order O for Σ';
(2) $Q^1 :=$ chase(Q, Σ', O);
(3) $Q^2 :=$ ans(Q^1);
(4) **return** Q^2.

Here we give a simplified version of FEASIBLE which gives an exponential time algorithm. This algorithm can be parallelized to give a $\mathbf{\Pi_2^P}$ algorithm when Q and Σ satisfy the assumptions of Theorem 7b, as outlined in the proof of that theorem.

function FEASIBLE(Q, Σ)
(1-3) *same as (1–3) of* REWRITE(Q, Σ);
(4) **if** $Q^2 =$ `undefined` **then return** `false`;
(5) $Q^3 :=$ chase(Q^2, Σ', O);
(6) **return** contained(Q^3, Q).

Theorem 6. *If $Q \sqsubseteq_\Sigma E$, E is executable, and Q^Σ is defined, then* ans(Q^Σ) *is defined and* $Q \sqsubseteq_\Sigma$ ans$(Q^\Sigma) \sqsubseteq E$.

Proof. Assume $Q \sqsubseteq_\Sigma E$ and E executable; by Theorem 4, $Q^\Sigma \sqsubseteq E$. Thus, by Theorem 2, $\text{ans}(Q^\Sigma)$ is defined and $Q^\Sigma \sqsubseteq \text{ans}(Q^\Sigma) \sqsubseteq E$. By Theorem 4, $Q \sqsubseteq_\Sigma \text{ans}(Q^\Sigma) \sqsubseteq E$.

Corollary 2. *The following are equivalent:*
1. *Q is Σ-feasible.*
2. *$\text{ans}(Q^\Sigma)$ is defined and $\text{ans}(Q^\Sigma) \sqsubseteq_\Sigma Q$.*
3. *$\text{ans}(Q^\Sigma)^\Sigma$ is defined and $\text{ans}(Q^\Sigma)^\Sigma \sqsubseteq Q$.*

Theorem 7. *a) If $\Sigma \subseteq \text{IC(UCQ)}$ and there is a polynomial p such that for all Q, $Q^\Sigma \in \text{UCQ}$: $|Q^\Sigma| \leqslant p(|Q|)$, then $\text{FEASIBLE}_\Sigma(\text{UCQ})$ is **NP**-complete.*
b) If $\Sigma \subseteq \text{IC}(\text{UCQ}^\neg)$ and there is a polynomial p such that for all Q, $Q^\Sigma (= \bigvee_i Q'_i) \in \text{UCQ}^\neg$ and for all i: $|Q'_i| \leqslant p(|Q|)$, then $\text{FEASIBLE}_\Sigma(\text{UCQ}^\neg)$ is $\mathbf{\Pi_2^P}$-complete.

The fact that Q^Σ is defined only up to equivalence is not a concern for our needs, due to the following result.

Lemma 2.
a) If $P \sqsubseteq Q$ and $\text{ans}(Q)$ is defined, then $\text{ans}(P)$ is defined and $\text{ans}(P) \sqsubseteq \text{ans}(Q)$.
b) If $P \equiv Q$ and $\text{ans}(Q)$ is defined, then $\text{ans}(P)$ is defined and $\text{ans}(P) \equiv \text{ans}(Q)$.

Proof. (a) If $\text{ans}(Q)$ is defined then it is executable and $P \sqsubseteq Q \sqsubseteq \text{ans}(Q)$. By Theorem 2, $\text{ans}(P)$ is defined and $\text{ans}(P) \sqsubseteq \text{ans}(Q)$. (b) follows from (a).

In general it is undecidable whether the chase terminates. [5] introduces a sufficient condition, checkable in **P**, for termination of the chase with IC(UCQ) constraints. It is fairly wide and generalizes the notions of full and acyclic dependencies [1]. The condition requires a set of constraints to have *stratified witnesses*.[2] We recall the definition in [3], where we extend the notion to sets of $\text{IC}(\text{UCQ}^\neg)$ constraints and also provide a proof of the following result.

Theorem 8. *For any $Q \in \text{UCQ}^\neg$, any $\Sigma \subseteq \text{IC}(\text{UCQ}^\neg)$ with stratified witnesses (this is checkable in **P**) and any total order O on Σ, the chase terminates. Moreover, Q^Σ satisfies the assumptions in Theorem 7.*

4 Views

We now consider the problem $\{Q, \mathcal{V}, \mathcal{P}, \Sigma_\text{c}\}$: given a query Q, a set of views $\mathcal{V}$ given as $\text{UCQ}^\neg$ queries $V_1, \ldots, V_n$ with access patterns $\mathcal{P}$ on the view heads, and a set of constraints Σ_c, we are interested in finding an executable Σ_c-rewriting of Q in terms of $V_1, \ldots, V_n$. That is, we want a query E over $\mathcal{V}$ that is Σ_c-equivalent to Q.

We reduce this case to the case of integrity constraints alone covered in the previous section as follows. Assume the views are over the schema τ. We can express the views as "forward" and "backward" constraints

$$\Sigma_\text{f}^\mathcal{V} := \{\forall \bar{x}_i \bar{y}_i \, (\text{body}(V_i) \to \text{head}(V_i)) \mid V_i \in \mathcal{V}, 1 \leqslant i \leqslant n\}$$

$$\Sigma_\text{b}^\mathcal{V} := \{\forall \bar{x}_i \, (\text{head}(V_i) \to \exists \bar{y}_i \text{body}(V_i)) \mid V_i \in \mathcal{V}, 1 \leqslant i \leqslant n\}$$

[2] The notion first arose in a conversation between the first author and Lucian Popa. It was then independently used in [8] under the term *weakly acyclic*.

over the schema $\tau \cup \tau_V$, where $\tau_V := \bigcup_i \text{head}(V_i)$ consists of all view heads; $\bar{x}_i$ are the variables in head(V_i), and $\bar{y}_i$ are the variables in body(V_i) which do not appear in head(V_i). Clearly, $\Sigma_{\text{f}}^{\mathcal{V}}, \Sigma_{\text{b}}^{\mathcal{V}} \subseteq \text{IC}(\text{UCQ}^{\neg})$. Set $\Sigma := \Sigma_{\text{c}} \cup \Sigma_{\text{f}}^{\mathcal{V}} \cup \Sigma_{\text{b}}^{\mathcal{V}}$.

The main results in this section are that we can reduce $\{Q, \mathcal{V}, \mathcal{P}, \Sigma_{\text{c}}\}$, the case of views with constraints, to $\{Q, \mathcal{P}, \Sigma\}$, the case without views via $\Sigma := \Sigma_{\text{c}} \cup \Sigma_{\text{f}}^{\mathcal{V}} \cup \Sigma_{\text{b}}^{\mathcal{V}}$, where the views $\mathcal{V}$ are captured by the constraints $\Sigma_{\text{f}}^{\mathcal{V}}$ and $\Sigma_{\text{b}}^{\mathcal{V}}$ above. In fact, it is enough to consider $Q^{\Sigma_{\text{c}}, \Sigma_{\text{f}}^{\mathcal{V}}}$ instead of Q^{Σ} for computing the answerable part (but for testing feasiblity we also need $\Sigma_{\text{b}}^{\mathcal{V}}$). If defined, $\text{ans}(Q^{\Sigma_{\text{c}}, \Sigma_{\text{f}}^{\mathcal{V}}}|\tau_V)$ is the minimal executable query over $\mathcal{V}$ Σ_{c}-containing Q (Theorem 9) where $Q|\tau$ is the query with the same head as Q and with body given by the literals in Q which have relation symbols in schema τ. It follows that the problem of whether there is a Σ_{c}-equivalent rewriting of a query Q over $\mathcal{V}$ can be reduced to checking containment (Corollary 3). We also show that we can stratify the chase and that we only need special conditions on Σ_{c} (but not on $\Sigma_{\text{f}}^{\mathcal{V}}$ or $\Sigma_{\text{b}}^{\mathcal{V}}$) to guarantee that Q^{Σ} is well-defined and suitably small (Theorem 10). We outline the algorithms ViewRewrite and ViewFeasible which use the functions ans(Q), chase(Q, Σ, O), and contained(P, Q).

function ViewRewrite($Q, \Sigma_{\text{c}}, \mathcal{V}$)
(1) Compute $\Sigma_{\text{c}}{}' := \Sigma_{\text{c}} \cup \Sigma_{\neg}^{\tau}$ and pick some order O_{c} for $\Sigma_{\text{c}}{}'$;
(2) Compute $\Sigma_{\text{f}}^{\mathcal{V}\prime} := \Sigma_{\text{f}}^{\mathcal{V}} \cup \Sigma_{\neg}^{\tau}$ and pick some order O_{f} for $\Sigma_{\text{f}}^{\mathcal{V}\prime}$;
(3) $Q^1 := \text{chase}(Q, \Sigma_{\text{c}}{}', O_{\text{c}})$;
(4) $Q^2 := \text{chase}(Q^1, \Sigma_{\text{f}}^{\mathcal{V}}, O_{\text{f}})$;
(5) $Q^3 := Q^2|\tau_V$ (that is, drop all τ literals);
(6) $Q^4 := \text{ans}(Q^3)$;
(7) **return** Q^4.

ViewRewrite($Q, \Sigma_{\text{c}}, \mathcal{V}$) may return `undefined` or may not terminate. Similarly, ViewFeasible($Q, \Sigma_{\text{c}}, \mathcal{V}$) may not terminate. Theorem 9 and Corollary 3 show that these algorithms are correct and complete, provided the chase terminates regardless of the order O. The simplified version of ViewFeasible below results in an exponential time algorithm; however, it can be parallelized to give a $\mathbf{\Pi_2^P}$ algorithm when Q and Σ satisfy the assumptions of Theorem 10.

function ViewFeasible($Q, \Sigma_{\text{c}}, \mathcal{V}$)
(1–6) *same as (1–6) of* ViewRewrite($Q, \Sigma_{\text{c}}, \mathcal{V}$);
(7) **if** $Q^4 =$ `undefined` **then return** `false`;
(8) Compute $\Sigma_{\text{b}}^{\mathcal{V}\prime} := \Sigma_{\text{b}}^{\mathcal{V}} \cup \Sigma_{\neg}^{\tau}$ and pick some order O_{b} for $\Sigma_{\text{b}}^{\mathcal{V}\prime}$;
(9) $Q^5 := \text{chase}(Q^4, \Sigma_{\text{b}}^{\mathcal{V}\prime}, O_{\text{b}})$;
(10) $Q^6 := \text{chase}(Q^5, \Sigma_{\text{c}}{}', O_{\text{c}})$;
(11) $Q^7 := Q^6|\tau$ (that is, drop all τ_V literals);
(12) **return** contained(Q^7, Q).

Theorem 9. *If defined,* $\text{ans}(Q^{\Sigma_{\text{c}}, \Sigma_{\text{f}}^{\mathcal{V}}}|\tau_V)$ *is the minimal executable query over* $\mathcal{V}$ Σ_{c}*-containing* Q *(otherwise there is no such executable query).*

Corollary 3. *There is an executable* Σ_{c}*-rewriting of* Q *over* $\mathcal{V}$ *iff* $\text{ans}(Q^{\Sigma_{\text{c}}, \Sigma_{\text{f}}^{\mathcal{V}}}|\tau_V)$ *is defined and* $\text{ans}(Q^{\Sigma_{\text{c}}, \Sigma_{\text{f}}^{\mathcal{V}}}|\tau_V)^{\Sigma_{\text{b}}^{\mathcal{V}}, \Sigma_{\text{c}}}|\tau \sqsubseteq Q$.

Theorem 10. *a) If* $\Sigma \subseteq$ IC(UCQ), $\mathcal{V} \subseteq$ UCQ *and there is a polynomial* p *such that for all* $Q, Q^{\Sigma_c} \in$ UCQ*:* $|Q^{\Sigma_c}| \leqslant p(|Q|)$*, then* ViewFeasible *is* **NP***-complete.*

b) If $\Sigma \subseteq$ IC(UCQ$^\neg$), $\mathcal{V} \subseteq$ UCQ$^\neg$ *and there is a polynomial* p *such that for all* Q, $Q^{\Sigma_c} (= \bigvee_i Q'_i) \in$ UCQ$^\neg$ *and all* i*:* $|Q'_i| \leqslant p(|Q|)$*, then* ViewFeasible *is* $\mathbf{\Pi_2^P}$*-complete.*

These results follow from the corresponding results in the previous section and the following considerations. Notice that Q' is a Σ_c-rewriting of Q over $\mathcal{V}$ iff $Q'|\tau_V = Q'$ and $Q' \equiv_\Sigma Q$ (the first part simply says that Q' is a query over $\mathcal{V}$). Since Σ includes the definitions of the views in $\mathcal{V}$, the second part expresses the desired equivalence under both Σ_c and the view definitions.

We know by Theorem 6 that $\text{ans}(Q^\Sigma|\tau_V)$ is the minimal executable query over τ_V Σ-containing Q. Since $\Sigma_f^\mathcal{V}$ and $\Sigma_b^\mathcal{V}$ express the equivalence of the views with their definitions over τ, this is the same as the minimal executable query over $\mathcal{V}$ Σ_c-containing Q. It follows that there is an executable Σ_c-rewriting of Q over $\mathcal{V}$ iff $\text{ans}(Q^\Sigma|\tau_V) \sqsubseteq_\Sigma Q$ iff $\text{ans}(Q^\Sigma|\tau_V)^\Sigma \sqsubseteq Q$.

The effect of chasing twice with Σ can be roughly described as follows:

- In Q^Σ we introduce the view heads.
- In $\text{ans}(Q^\Sigma|\tau_V)$ we remove the original literals in Q and the view bodies.
- In $\text{ans}(Q^\Sigma|\tau_V)^\Sigma$ we expand the view heads to again include their bodies.

At this point, we have a query over $\tau \cup \tau_V$, but since Q is over τ, only the τ part matters. Therefore $\text{ans}(Q^\Sigma|\tau_V)^\Sigma \sqsubseteq Q$ iff $\text{ans}(Q^\Sigma|\tau_V)^\Sigma|\tau \sqsubseteq Q$. Furthermore, $\text{ans}(Q^\Sigma|\tau_V)^\Sigma|\tau \sqsubseteq Q$ iff $\text{ans}(Q^{\Sigma_c,\Sigma_f^\mathcal{V}}|\tau_V)^{\Sigma_b^\mathcal{V},\Sigma_c}|\tau \sqsubseteq Q$ since $Q^{\Sigma_c,\Sigma_f^\mathcal{V},\Sigma_b^\mathcal{V}} \equiv Q^{\Sigma_c,\Sigma_f^\mathcal{V}}$. This is because $\Sigma_f^\mathcal{V}$ only introduces atoms with relation symbols from τ_V and these in turn can only "fire" constraints from $\Sigma_b^\mathcal{V}$ which reintroduce bodies that have already been matched (with new quantified variables). Such chase steps never apply.

The chases with $\Sigma_f^\mathcal{V}$ and $\Sigma_b^\mathcal{V}$ can be done in a one step since new atoms added by $\Sigma_f^\mathcal{V}$ constraints have relation symbols in τ_V, whereas $\Sigma_f^\mathcal{V}$ constraints must match atoms with relation symbols in τ; similarly for $\Sigma_b^\mathcal{V}$ with the roles of τ and τ_V reversed. Only chasing with Σ_c may result in an infinite chase or in Q being too large.

Notice that, by Theorem 8, we can test in polynomial time whether Σ_c meets sufficient (and fairly wide) conditions that satisfy the assumptions of Theorem 10.

5 Maximal Contained Rewritings

Since exact rewritings of a query Q do not always exist, we want to approximate Q as best as possible. In Sections 3 and 4 we have shown how to obtain the minimal containing rewritings, which are the best overestimates of Q. In this section we consider maximal contained rewritings of Q, which are the best underestimates of Q.

Given a schema τ, let D_τ be the unary recursive query given by rules of the form $D_\tau(x_j) \leftarrow D_\tau(x_{i_1}), \ldots, D_\tau(x_{i_k}), R(\bar{x})$ for every relation $R \in \tau$ and every access pattern R^α where $x_{i_1}, \ldots, x_{i_k}$ are the input slots of R^α and j is an output slot in R^α.

Definition 7 (Domain Extension). The *domain extension* of $Q \in \text{CQ}^\neg$ is another query $\text{dext}(Q)$ given by the rules with head $D_\tau(x_j)$ mentioned above and the rule

$$\text{dext}(Q)(\bar{x}) \leftarrow D_\tau(y_1), \ldots, D_\tau(y_k), \text{body}(Q)$$

where y_i are the variables in $\text{body}(Q)$.

For $Q(= \bigvee_i Q_i) \in \text{UCQ}^\neg$, we define $\text{dext}(Q) := \bigvee_i \text{dext}(Q_i)$.

Notice that D_τ and $\text{dext}(Q)$ are recursive queries; in particular, here we deviate from the convention in Section 2 that all the rules of a query have the same head. Clearly, $\text{dext}(Q)$ is executable.[3] D_τ, $\text{dext}(Q)$, and the following result are given in [7] for CQ.

Theorem 11. *If $E \sqsubseteq Q$, E is executable, and E contains no constants, then $E \sqsubseteq \text{dext}(Q) \sqsubseteq Q$.*

We must disallow constants since they can be used to partially enumerate the domain. If we allow constants and '=', we can add rules of the form $D_\tau(x) \leftarrow (x = c)$ for every constant c. Notice that nothing special needs to be done here to handle negation since negative literals do not contribute towards enumerating the domain.

Theorem 12. *If $E \sqsubseteq_\Sigma Q$, E is executable, E contains no constants, and E^Σ is defined, then $E \sqsubseteq_\Sigma \text{dext}(Q) \sqsubseteq Q$.*

Proof. If E is executable, then E^Σ is also executable and satisfies $E^\Sigma \sqsubseteq Q$. Therefore, $E^\Sigma \sqsubseteq \text{dext}(Q)$ holds, which is equivalent to $E \sqsubseteq_\Sigma \text{dext}(Q)$.

Now assume that as in Section 4 we have a query Q, a set of constraints Σ_c, and a set of views $\mathcal{V}$ given by $\text{UCQ}^\neg$ queries $V_1, \ldots, V_n$ with access patterns on the heads of the views. We express the views as constraints $\Sigma_\text{f}^\mathcal{V}$ and $\Sigma_\text{b}^\mathcal{V}$ as in Section 4. We are interested in finding a maximal Σ_c-contained executable rewriting of Q in terms of $V_1, \ldots, V_n$. That is, we want a query over $\mathcal{V}$ that is maximally Σ-contained in Q.

Theorem 13. *If E is a maximal Σ_c-contained rewriting of Q over $\mathcal{V}$ (regardless of access patterns), then $\text{dext}(E)$ is a maximal contained executable Σ_c-rewriting of Q.*

Proof. Assume E is as in premise and P is an executable query over $\mathcal{V}$ and $P \sqsubseteq_\Sigma Q$. Then $P \sqsubseteq_\Sigma E$ by the maximality of E. Since P is executable, by Theorem 12, $P \sqsubseteq_\Sigma \text{dext}(E)$.

[7] shows how to compute such a maximal Σ_c-contained rewriting of Q in the absence of negation using a recursive plan. But it is easy to see that such recursive plans can be transformed into a union of conjunctive queries: we simply take the union of all minimal CQ queries over $\mathcal{V}$ which are Σ_c-contained in Q (the results of [7] imply that this union is finite when the chase terminates). The extension to handle negation is straightforward and we omit it in view of our results in the next section.

[3] We have not defined "executable" for recursive queries, but the extension is straightforward.

6 Reducing Access Patterns to Constraints

In this section, we show that the problem $\{Q, \mathcal{P}, \Sigma\}$ of deciding feasibility in the presence of access patterns reduces to the problem $\{Q, \Sigma\}$ of deciding equivalence in the presence of constraints only (Theorem 14). Furthermore, we reduce the problem $\{Q, \mathcal{V}, \mathcal{P}, \Sigma\}$ of finding rewritings using views with access patterns to one of finding rewritings using views and constraints in the absence of access patterns $\{Q, \mathcal{V}, \Sigma\}$ (Theorem 15). These results enable alternative proofs for the complexity of answering queries in the presence of access patterns. They also facilitate an alternative implementation of algorithms Rewrite, Feasible, ViewRewrite etc. using a chase-based module for rewriting under constraints such as the **C&B** implementation in [4].

The reduction is based on the observation that the domain enumeration program D_τ from Section 5 is a view (albeit recursive) and can therefore be captured with integrity constraints, as shown in Section 4. Call the set of resulting constraints Σ_D; notice that $\Sigma_D \subseteq \mathrm{IC}(\mathrm{UCQ}^\neg)$.

Theorem 14. *Q is Σ-feasible iff $Q \sqsubseteq_{\Sigma_D \cup \Sigma} \mathrm{dext}(Q)$.*

Since $\mathrm{dext}(Q) \sqsubseteq Q$, we only need to check $Q \sqsubseteq_{\Sigma_D} \mathrm{dext}(Q)$ in Theorem 14. We thus retrieve the complexity results for checking feasibility from the complexity of checking containment. If $\Sigma = \emptyset$ we can show that (a) $\mathrm{chase}(Q, \Sigma_D, O)$ terminates for any order O on Σ_D, (b) the chase result is unique regardless of O (denote it Q^{Σ_D}), and (c) if we restrict Q^{Σ_D} to only those atoms $R(\bar{x})$ for which $D(\bar{x})$ appears in Q^{Σ_D}, we obtain $\mathrm{ans}(Q)$. This and Theorem 15 below enable an alternative implementation of algorithm ViewRewrite, which does not compute $\mathrm{ans}(Q^3)$ in step (6) but instead uses $\mathcal{V}^D$ for $\mathcal{V}$ and $\Sigma \cup \Sigma_D$ for Σ.

We now reduce the rewriting problem $\{Q, \mathcal{V}, \mathcal{P}, \Sigma\}$ to $\{Q, \mathcal{V}, \Sigma\}$. First, define D_τ as in Section 5, but using view symbols instead of relation symbols from τ. Next capture D_τ with constraints Σ_D. For any $V \in \mathcal{V}$ and access pattern V^α, define a new view $V^D(\mathrm{free}(V)) :- \mathrm{body}(V), D(x_{i_1}), \ldots, D(x_{i_k})$ where the $x_{i_j} \in \mathrm{free}(V)$ are the free variables of V which appear in input slots. Each V^D is a view without access patterns. Denoting $\mathcal{V}^D := \{V^D \mid V \in \mathcal{V}\}$, we have the following result.

Theorem 15. *a) Q has an executable Σ-rewriting over $\mathcal{V}$ iff it has a $\Sigma \cup \Sigma_D$-rewriting over $\mathcal{V}^D$.*

b) For each exact (minimal containing) $\Sigma \cup \Sigma_D$-rewriting of Q over $\mathcal{V}^D$, we can derive in polynomial time an exact (minimal containing) Σ-rewriting of Q over $\mathcal{V}$.

7 Extensions

The key technique that allows us to treat negation, views, and access patterns uniformly is modeling with constraints (recall $\Sigma^\tau_\neg$, $\Sigma^{\mathcal{V}}_{\mathrm{f}} \cup \Sigma^{\mathcal{V}}_{\mathrm{b}}$, respectively Σ_D). This approach enables the straightforward implementation of our algorithms by reusing an already existing chase module [4]. It turns out that we can extend our

solution to handling equality and arithmetic comparisons by capturing them with constraints as well.

Handling Equality. Equality can be modeled as a binary relation E with access patterns 'io' and 'oi' subject to the following constraints $\Sigma_=^\tau \subseteq \text{IC(CQ)}$:

- $\forall x \ \texttt{true} \rightarrow E(x,x)$,
- $\forall x,y \ \ E(x,y) \rightarrow E(y,x)$, and,
- for every $R \in \tau$: $\ \forall \bar{x},\bar{y} \ \ R(\bar{x}) \wedge E(x_1,y_1) \wedge \ldots \wedge E(x_k,y_k) \rightarrow R(\bar{y})$.

Handling Arithmetic Comparisons. The comparison '$\leq$', which gives UCQAC$^\neg$, can be handled as a binary relation LE with access pattern 'ii' subject to the following constraints $\Sigma_\leq \subseteq \text{IC(CQ}^\neg)$ which say that LE is an unbounded dense total ordering:

- $\forall x,y,z \ \ LE(x,y) \wedge LE(y,z) \rightarrow LE(x,z)$,
- $\forall x,y \ \ LE(x,y) \wedge LE(y,x) \rightarrow E(x,y)$,
- $\forall x,y \ \ \neg LE(x,y) \rightarrow LE(y,x)$, and
- $\forall x,y \ \ L(x,y) \rightarrow \exists u,v,w \ (L(u,x) \wedge L(x,v) \wedge L(v,y) \wedge L(y,w))$,

where $L(x,y)$ stands for $LE(x,y) \wedge \neg E(x,y)$.

Notice that the chase with the last axiom (the density axiom) is non-terminating, yielding chains of $<$ comparisons of arbitrary length. However, we can show that if in each integrity constraint all variables that appear in a $\leq$ atom also appear in some relational atom other than a $\leq$ atom, then there is no need to chase with the density axiom. In this case, all of our results extend to unions of conjunctive queries with negation, equality and arithmetic comparisons as well as the corresponding constraints. All we need to do is replace Σ with $\Sigma' := \Sigma \cup \Sigma_\leq \cup \Sigma_=^\tau$ and run algorithms Feasible, ViewRewrite, Rewrite, ViewFeasible on Σ'.

Even if the restriction above does not hold, it can be shown that the chase with the density axiom can be truncated so as to generate $<$ chains of length bounded by the number of variables in the original query. All we need to do is run algorithms Feasible,ViewRewrite,Rewrite,ViewFeasible using the truncating chase.

References

1. Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison Wesley, 1995.
2. A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *ACM Symposium on Theory of Computing (STOC)*, pages 77–90, 1977.
3. Alin Deutsch, Bertram Ludäscher, and Alan Nash. Rewriting queries using views with access patterns under integrity constraints. Technical report, Department of Computer Science and Engineering, UCSD, 2004. extended version of this paper: http://www.db.ucsd.edu:8080/root/index.jsp?pageStr=publications.
4. Alin Deutsch and Val Tannen. Mars: A system for publishing xml from mixed and redundant storage. In *Intl. Conf. on Very Large Data Bases (VLDB)*, 2003.
5. Alin Deutsch and Val Tannen. Reformulation of xml queries and constraints. In *Intl. Conf. on Database Theory (ICDT)*, 2003.

6. Oliver M. Duschka and Michael R. Genesereth. Answering recursive queries using views. In *PODS*, 1997.
7. Oliver M. Duschka, Michael R. Genesereth, and Alon Y. Levy. Recursive query plans for data integration. *Journal of Logic Programming*, 43(1):49–73, 2000.
8. Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: Semantics and query answering. In *Intl. Conf. on Database Theory (ICDT)*, 2003.
9. Daniela Florescu, Alon Y. Levy, Ioana Manolescu, and Dan Suciu. Query optimization in the presence of limited access patterns. In *SIGMOD*, pages 311–322, 1999.
10. J. Grant and J. Minker. A logic-based approach to data integration. *Theory and Practice of Logic Programming*, 2(3):323–368, 2002.
11. Alon Halevy. Answering queries using views: A survey. *VLDB Journal*, 10(4):270–294, 2001.
12. Christoph Koch. Query rewriting with symmetric constraints. *AI Communications*, 17(2), 2004. to appear.
13. Alon Y. Levy. Logic-based techniques in data integration. In Jack Minker, editor, *Workshop on Logic-Based Artificial Intelligence, Washington, DC*, 1999.
14. Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Querying heterogeneous information sources using source descriptions. In *22nd Intl. Conf. on Very Large Data Bases (VLDB)*, pages 251–262, Bombay, India, 1996.
15. Chen Li. Computing complete answers to queries in the presence of limited access patterns. *Journal of VLDB*, 12:211–227, 2003.
16. Chen Li and Edward Y. Chang. On answering queries in the presence of limited access patterns. In *Intl. Conference on Database Theory (ICDT)*, 2001.
17. Todd D. Millstein, Alon Y. Levy, and Marc Friedman. Query containment for data integration systems. In *PODS*, pages 67–75, 2000.
18. Alan Nash and Bertram Ludäscher. Processing first-order queries under limited access patterns. In *PODS*, Paris, France, 2004.
19. Alan Nash and Bertram Ludäscher. Processing unions of conjunctive queries with negation under limited access patterns. In *Intl. Conference on Extending Database Technology (EDBT)*, Heraklion, Crete, Greece, 2004.
20. A. Rajaraman, Y. Sagiv, and J. D. Ullman. Answering queries using templates with binding patterns. In *PODS*, pages 105–112, 1995.
21. Yehoshua Sagiv and Mihalis Yannakakis. Equivalences among relational expressions with the union and difference operators. *Journal of the ACM*, 27(4):633–655, 1980.
22. Jeffrey Ullman. The complexity of ordering subgoals. In *PODS*, 1988.
23. Vasilis Vassalos and Yannis Papakonstantinou. Expressive capabilities description languages and query rewriting algorithms. *Journal of Logic Programming*, 43(1):75–122, 2000.